



Project 1

Prof. Adín Ramírez Rivera
adin@ic.unicamp.br

1 Description

Image compositing and blending is a topic that has been studied widely by the Computer Vision Community. Nowadays, several software implement this functionalities automatically, and has become widespread due to its ease of use. The compositing operation consists on introducing parts of one image into another, and seamlessly blend them together.

In this project you will be doing image blending through the spatial pyramids and convolutional operations. On the second part you will explore the frequency domain for blending as well.

Your assignment is to create basic operations, such as convolution operation, pyramid handling, up and downscale, and masking, and use them to create a blending function. For the frequency part, you are allowed to use the OpenCV functions for Fourier transformation.

2 Spatial Blending

The spatial blending that you will be implementing is inspired by the seminal work of Burt and Adelson [1]. In their work, a spatial pyramid of Laplacians is constructed, and then at the lower level, the images are blended. Then, the high resolution image is recovered by reconstructing the image through the pyramid. In this part of the work you will create the operations to perform such operation.

A pyramid is a data structure that holds a set of images. Commonly, the highest level holds the lower scale image, while the highest one the original size. The pyramid is then constructed based on the number of levels and in each level the image is downsized by a factor of two. A Laplacian pyramid is created by creating a Gaussian pyramid and then subtracting the reconstructed level with the original image. Hence, in this project you need to create your own pyramid abstraction, using your own operators.


2.1 Convolutions

You need to create a `convolution` function that receives an input image and a convolution mask, and that returns the convolved image as output. A suggested signature¹ for the function is

```
void convolution(InputArray input, OutputArray output, InputArray mask).
```

Your `convolution` function should perform a general convolution operation, and return an image of the same size as `input` (that is, make no assumptions about `mask`). In your report explain your approach to handle borders, and justify it.

Additionally, make a couple of experiments with some masks (for example test 3×3 , 7×7 , 15×15 masks) and measure the execution time on your function. Repeat the experiment with the convolution function provided by OpenCV.

 In your report, show and explain your findings, and justify your results. There is no need to show all the images (a couple of them will suffice), instead show a graph or table with the comparison and explain your findings.

¹In this project the signatures will be given in C++ for brevity. You can infer the C or Python equivalent if needed.

2.2 Gaussian Pyramid

You need to create a data structure to handle your Gaussian pyramid. This pyramid is constructed using an input image, and a number of levels. The construction process is as follows. Blur the image of a given level using a Gaussian mask (explain your choice of mask size or experiment with different sets of masks and show your results), and then down-sample it by removing every other row and column. The image of the next level is the result of the previous operation. Repeat this process for all the levels. The first image is the original input image. Note that since you are removing every other row and column during the down-sampling operation, you are reducing the size of the image by a factor of two.

For the reconstruction process of a given level, you need to perform the inverse operation. First, you duplicate the size of the level and insert the existing pixels in every other row and column. The missing information should be interpolated. You can use bilinear interpolation (✍️ if you want you can test other types of interpolation and explain them in your report).

Your data structure should implement these two operations with two functions: `up` and `down`, that receive an image and return the corresponding image. Note that since we define our original image to be the base level of the pyramid, we will construct it using the `up` function, while we will recover the previous images by using the `down` function.

Additionally, add an access function to obtain a given level of your pyramid when needed. ✍️ Explain your design decisions on the implementation of the data structure and reason why.

2.3 Laplacian Pyramid

Similarly to the Gaussian pyramid, you will construct a Laplacian pyramid data structure with similar functionality. The same idea of `up` and `down` functions apply.

The construction of a level of the Laplacian pyramid is as follows (`up` operation). Let G be the corresponding Gaussian pyramid, of the given image. At level i , you need to reconstruct the next level of the Gaussian (e.g., `G[i].down()` or `Gaussian::down(G[i])` depending on your definition of the previous section) and subtract it from the current level image (in the Laplacian pyramid). The result is the current Laplacian level.

The reconstruction (`down` operation) of a given level does the opposite operation (that is, it reverses the operations to produce the original image).

✍️ Explain in your report how do you perform the reconstruction in detail. Additionally, explain your design decisions on the implementation of the data structure and reason why.

2.4 Blending

Using your implementation of the Gaussian and Laplacian pyramids you need to implement the blending operation proposed by Burt and Adelson [1]. That is, you need a pair of images and a blending mask, of the same sizes. The blending mask is an image of the same size as the inputs that contains 1's (or 255's in case of 8 bit images) that marks the interest region. Construct your Gaussian and Laplacian pyramids of a given pair of images, and the Gaussian pyramid of the mask.

At the highest level of the Laplacian pyramids, blend the corresponding images using the mask. The first image should use the mask as is, and the second one should use its complement. Then, reconstruct the resulting image with the `down` operation of the Laplacian pyramid data structure.

✍️ Explain your process in your report and show your results. Try to reproduce the results presented by the original authors [1]. Test your algorithm with several images and different masks (include such images in the input folder). Explain the limitations of your algorithm and show cases in which it fails. Can you explain why?

3 Frequency Blending

After your experiments with blending images on the spatial domain, we will work with the frequency domain.

3.1 Exploring Fourier Space


Before doing the blending you will perform some experiments with the frequency domain. Pick an image and convert it to the frequency domain (you can use OpenCV Fourier transform operations for this).

You should have a magnitude and a phase image. Select the phase image and select the lowest value (bigger than zero) in that image. Create a new phase image that contains only the selected pixels. And use it to reconstruct an image. Save your result.

Reproduce this step, with the difference that you add more points per iteration. Show your results at the 25%, 50%, 75%, and 100% of selected points. You should have five images (one with one point, and four with the given percentages of points).

Repeat this process with the phase image, but select the points in decreasing order. That is, select first the highest frequency, and then include the percentages of the top 25%, and so on. You should have five images as well.


Repeat the same process of decreasing and increasing orders and selection for the magnitude image. I advice you to create a function that builds this functionality given an image and a number or percentage of points to include.

 Show and explain in your report what your results are. Does the magnitude and the phase have the same importance? Explain your answer.

3.2 Blending

Using the same images (input, output, and mask) as in § 2.4, mask the images. That is, you should have information on the corresponding part of the mask, and zero elsewhere. Note that your mask need to be inverted for one image (similar to the process of the blending in the spatial domain).

Convert the masked images to the frequency domain. Now, you need to blend them together. And convert the blended frequencies into an image.

Experiment with different blending methods of the phase and magnitude. Use your experience from the previous section.  In your report, explain your proposed blending method, and why it works (or doesn't). Show results comparing the spatial blending with the frequency blending.

4 Evaluation

Your grade will be defined by the following aspects:

1. Convolution experiments	20%
2. Gaussian Pyramid design and implementation	3%
3. Laplacian Pyramid design and implementation	7%
4. Spatial blending experiments	20%
5. Frequency domain experiments	20%
6. Frequency blending experiments	20%
7. Overall report	10%

Each item corresponds to the questions and requirements defined in the previous sections. The overall report, point 7, refers to the evaluation of the details presented in your report, your way of presenting results, explanation and use of concepts and theory, references, etc. **Your English usage won't be graded**, but your ability to present your results, ideas, and how they are supported will be. Each other point will be evaluated according to the completeness and correctness of the requested items.

5 Submission

You need to create a folder named p1-XX-YY where XX and YY are the RAs of the members of your team. Note that the RAs **must** be sorted.

Your submission must have the following subfolders:

- **input**: a directory containing the input assets (images, videos or other data) supplied with the project. **Store the images used to produce the experiments here.** For any other image, setup your Makefile to automatically download them from a public server or repository.

- **output**: a directory where your application should produce all the generated files (otherwise stated in the problem).

In case of a problem that asks for producing images automatically, the convention for naming the images must be `p<project #>-<question #>-<part>-<counter>.png`. Note that `<part>` may be omitted if the question does not have one, and that `<counter>` must be autoincremented starting from 0.

- **src**: a directory containing all your source code. You only need to submit files that are not derivated from other files or through compilation. In case some processing is needed, prefer to submit a script that does that instead of submitting the files.
- **Makefile**: a makefile that executes your code through the docker image. An image is already built and available for use (`adnrv/opencv` at the docker hub registry). The code will be executed through a standard call to `make`, so other dependencies must be provided by you under that constraints.
- **report.pdf**: a PDF file that shows all your work for the given project, including images (labeled appropriately, that is, in accordance to the convention given) and other outputs needed to explain and convey your work. When needed include explanations to the questions given in the project.

The principal folder must be zipped into `p1-XX-YY.zip` and submitted through the Moodle website. No other files will be accepted.

6 Notes

- ❗ All the submissions must be self contained and must be executable in a Linux environment. Specifically, your code must execute in the docker image `adnrv/opencv`, available at docker hub (<https://hub.docker.com/r/adnrv/opencv/>).
- ❗ It is your responsibility to make sure your code compiles and executes correctly. No effort will be made to run your code besides executing `make` inside a docker.
- ❗ You must program in Python 3.6 (or higher) or in C/C++ with `gcc` version 6.2.0, using OpenCV 3.2.0. All available within the image. If you need to install other packages you must do so within your Makefile as automatic prerequisites.
- ❗ Check the Makefile provided with the project zero. It will be used to automatically execute your code. You must pass the full path of your project folder: `make SOURCE_DIR=$(realpath ./pA-XX-YY)`.

References

- [1] P. J. Burt and E. H. Adelson, "A multiresolution spline with application to image mosaics," *ACM Transactions on Graphics (TOG)*, vol. 2, no. 4, pp. 217–236, 1983. [Online]. Available: http://persci.mit.edu/pub_pdfs/spline83.pdf.