ECE558 Project03-Final

Note: Please do NOT search online for potential solutions. You are encouraged to discuss ideas and problems with your classmates or friends and/or instructor and TA when needed. Only by doing so, the homework and projects could help you best. By the end of this course, what you really learn and understand matter the most, not the gradings themselves.

How to submit your solutions: put your report (word or pdf) and results images (please use .png format) in a folder named [your_unityid]_hw01 (e.g., twu19_hw01), and then compress it as a zip file (e.g., twu19_hw01.zip). Submit the zip file through moodle.

- We will **NOT** accept any replacement of submission after deadline, even if you can show the time stamp of the replacement is earlier than the deadline. So, please double-check if you submit correct file(s).
- If you use your late day(s), please submit your solutions via email to both the Instructor and TA.

Important Note: No late days for the final project due to the deadline of submitting the final grades to the university office.

Teaming [Optional]: You can form a team with no more than 3 members in total to work on one of the two options as follows. The contributions from each team member need to be clearly presented in one separate section of your report. You can also work on the project individually.

It is also a good time to learn from each other and work together to improve the codes in project01/02 to be re-used /updated in this project. You can share and merge your codes in project01/02 within the team ONLY.

5-point Bonus: It will be given to the top-3 projects. If it is a team project, all the members will get the bonus. The projects will be ranked using the following criteria:

- The code is clean and self-contained. It can run without extra packages unless clean and clear installation scripts are provided. Basically, we need to 1-click code to test the results.
- The code is fast. We will compare the average runtime for some test images (held out).
- The report gives details of how you implement the code, how you address different issues.
- If you are not interested in competing for the 5 points, please make it clear in your report.
- If you are indeed interested in this, it may be a good opportunity to reshape all the codes you have implemented in this class. Be proud of your achievement by running your code faster in a cleaner way!

Option 1:

Description **(100 points)**: Pyramid Blending.

(a) [40 points] Write a function to implement Gaussian and Laplacian pyramid,

$$gPyr,\ lPyr = ComputePyr(input\_image,\ num\_layers),$$

Input arguments (for your reference): $input\_image$ is an input image (grey, or RGB), num_layers is the number of layers of the pyramid to be computed. Depending on the size of $input\_image$, num_layers needs to be checked if valid. If not, use the maximum value allowed in terms of the size of $input\_image$.
Outputs: $gPyr,\ lPyr$ are the Gaussian pyramid and Laplacian pyramid respectively.

1) The smoothing function: you can use built-in functions to generate the Gaussian kernel (think about and research what the kernel size should be, as well as [optional] ablation studies in your experiments), but you need to use your own conv function or FFT function implemented in Project 2. It's a good time to improve your conv and/or FFT implementation as you see fit.
2) The downsampler (for GaussianPyr) and upsampler (for LaplacianPyr) use the simplest nearest neighbor interpolation.

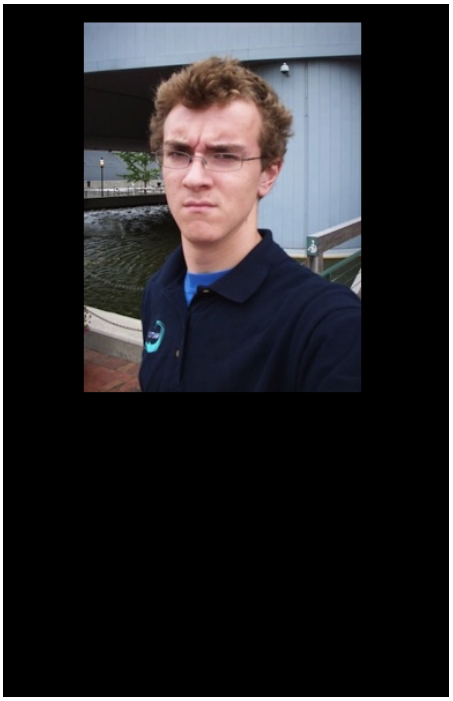
(b) [20 points] Write a simple GUI to create a black/white binary mask image. The GUI can open an image (e.g. the foreground image that you will use in blending); On the image, you can select a region of interest using either a rectangle or an eclipse, [optional] even some free-form region. Based on the opened image and the selected regions, the GUI can generate a black/white mask image of the same size as the opened image, in which the selected region(s) are white and the remaining black.

Note: *For this GUI, you can search online and reuse whatever functions you find useful and can be put together as* **a single self-contained module to realize the aforementioned mask generation functionality. But, you need to finish this on your own.**

(c) [40 points] On top of the functions in (a) and (b), write a function to implement Laplacian pyramid blending (see lecture note 19).

1) Provide at least 3 pairs of images. You can search images online to find interesting pairs that will generate fun blending results. Be creative and have fun! **Note: it will be of low probabilities that the same image pairs will be used by many of you, even you have a same pair, you will have different region of interest generated by the GUI, so we will expect to see different pairs of images and their blending results in your results.**

Example:

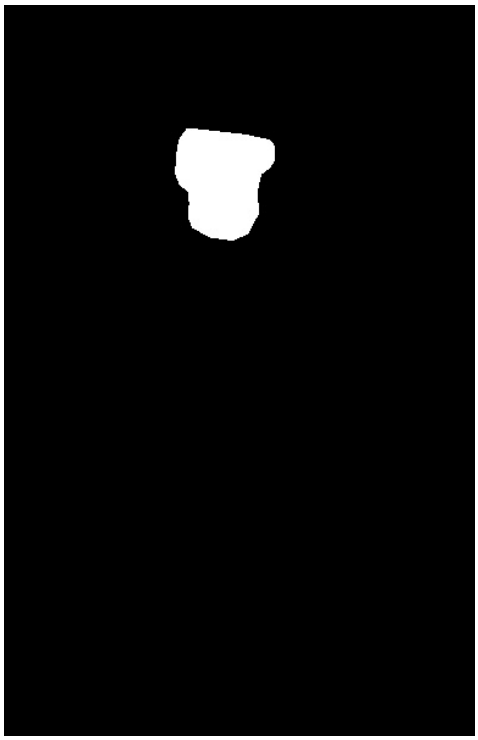


Foreground/source image

Background/target image

The foreground image can be your own photo if you do not mind. Here we want to blending the source face into the target face, so we align these two by placing the "actual" source image in a black background. For scenarios like this, you need to write a function to implement this: first generate a black background image of the same size as the target image, and then place the "actual" small source image at a proper location. The location can be manually aligned and tuned and then use as input argument for the function.

With the foreground image, use the GUI to generate a mask, and the do the blending.



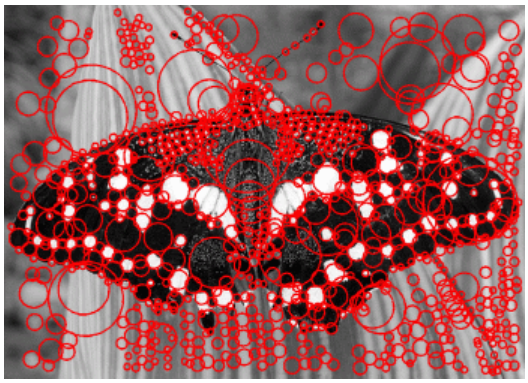
Mask

Blending result (illustration)

Option 2:

Project description: As illustrated by the below two images, the objective of this project is to implement a Laplacian blob detector.

An input image



A blob detection result

Algorithm outline:

- Generate a Laplacian of Gaussian filter.
- Build a Laplacian scale space (you need to use your own conv function or FFT function implemented in Project 2. It's a good time to improve your conv and/or FFT implementation as you see fit.), starting with some initial scale and going for n iterations:
    - Filter image (you need to use your own conv function or FFT function implemented in Project 2. It's a good time to improve your conv and/or FFT implementation as you see fit.) with scale-normalized Laplacian at current scale.
    - Save square of Laplacian response for current level of scale space.
    - Increase scale by a factor k.
- Perform non-maximum suppression (core function to be implemented on your own, new for project3) in scale space.
- Display resulting circles at their characteristic scales.

Test images: four images are provided and the testing results are required. Note that your output may look different depending on your threshold, range of scales, and other implementation details. In addition to the images provided, **also run your code on at least four images of your own choosing (required)**.

Requirement: Your report need to provide details of how you elaborate the algorithm outline and how you implement them. Your code should be self-contained: Given an input RGB image, it will generate the detection results (see the example above). Your code can be run without extra packages unless clear installation instructions are provided step-by-step. You need to implement the entire algorithm independently without built-in functions used for core components, except for the image I/O and displaying functions. E.g. You can reuse your convolution code.