ENGR 13300

Python Group Project Fall 2023

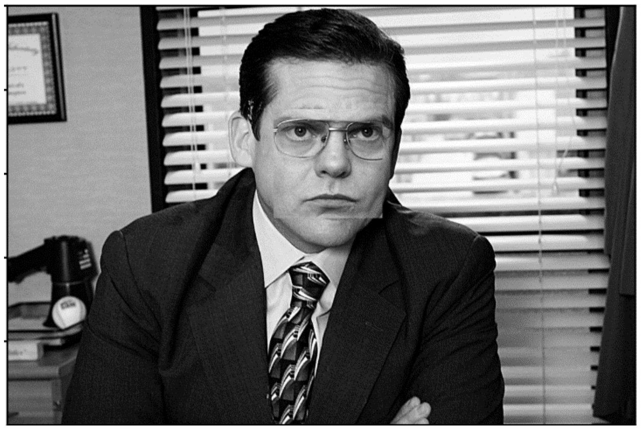
Project Description

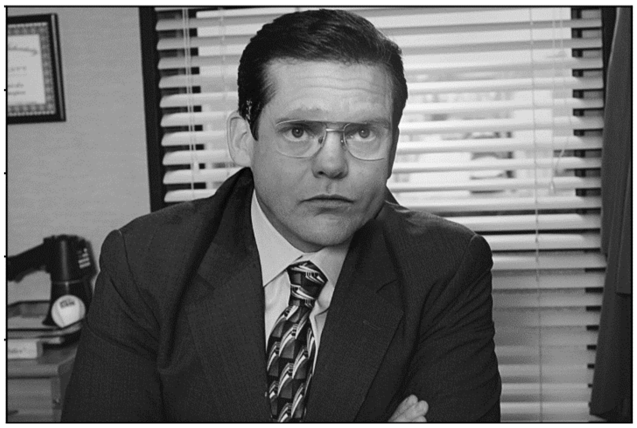
Image processing tools are becoming ever more powerful and prevalent in our world. Whether in computer graphics, facial recognition, object detection, image restoration, or deep fakes, the ability to easily harness greater and greater amounts of processing power has led to easy access to image processing techniques.

In this project, you are tasked with smoothly combining two images by implementing a method called Pyramid Blending. This technique uses “pyramids,” which are a collection of images at different resolutions, to smoothly combine two images. Your will implement this method for a pair of images that are provided and for a pair of images of your choosing.

Blending with Image Pyramids

Utilizing image pyramids is one method of blending images smoothly, and the method you will implement in this project. The images below show what a simple cut-and-paste method would produce and what image blending using pyramids looks like.


A simple cut-and-paste of one image onto another.


Two images combined using pyramid blending.

Here is an outline of the process. More detail of each step is provided further on. See the “Full Algorithm Visual Representation” pdf for a visual depiction of the process described below.

1. **Create the needed images and masks**
    a. Apply a Gaussian filter to the source and target images and their masks, which causes them to be blurred.
    b. The blurred images and masks are then subsampled, reducing their size by a factor of two.
        i. *Note: It is important to keep track of the dimensions of each image and mask you create in this process. To use the masks or add together the images, the arrays need to be the same size.*
    c. The smaller images from step 1b are then upsampled, increasing their size by a factor of two, and processed with the Gaussian filter again.

   d. Subtract the image arrays created in step 1c from the original image array for both the source and target images. (This creates images that contains only the "details" from the original image). These are called the **Laplacian Images**.
   e. Multiply each image array in step 1d by its corresponding mask.
   f. Add the images arrays from 1e together to create a combined Laplacian image.
   g. Repeat steps a-f as needed to create the desired number of pyramid levels.
      i. *Note: You will need to store each image and mask that you create in some kind of list, array, dictionary, etc.*
2. **Put the pieces together to create a combined image**
   a. Take the smallest upsampled images (which was created the last time you did step 1c) and apply the appropriate mask to each image.
   b. Add the images from 2a together to create a combined image.
      i. Remember, images are just arrays, so you can perform operations with them like you would any other array.
      ii. Make sure you appropriately place the source on the target.
   c. Add the image from 2b with appropriate Laplacian image from 1f.
   d. Upsample the image from 2c.
   e. Add the appropriate Laplacian image from 1f.
   f. Repeat steps d-e as needed to produce an image that is the same size as the original target image.

See the "Visual Algorithm Representation" document for a visual breakdown of the process outlined above.

Requirements

1. Your team is only allowed to import from `NumPy`, `matplotlib`, and the Python standard library in your final submission of this project.
2. Utilize functions in this project to allow for modularity and reusability. This will help with the division of labor in your team and make the project easier.

Deadlines

- In class demonstration: on or before October 19th.
- Final reports and code submissions uploaded to Gradescope by the end of class on October 19th.

Overview

To complete this project your program will need to do all of the following, but not necessarily in the order given below.

- Get the user inputs
- Open color images as arrays
- Convert color images to grayscale
- Create image masks
- Blur and subsample images
- Create a Gaussian image pyramid

- Upsample images
- Create a Laplacian image pyramid
- Utilize the image pyramids to reconstruct the combined image
- Display the image outputs

Each of these are described below in more detail. As you read through each item consider the following: Which elements should go in the main program? Which should be user-defined functions? Where might loops be useful? Which parts does your team need to program and which parts do NumPy or matplotlib have built-in functions for?

Algorithm explanation

Each section below explains what your team's program needs to do. Use user-defined functions where appropriate. These should *not* all be in your `main` program.

Getting User Input

Your program should prompt the user for the following inputs:

1. The name of a color **source** image file. (This is the image you will "cut" from.)
2. The name of a color **target** image file. (This is the image you will "paste" to.)
    a. *Hint: It acceptable to hard code the image file names while you are testing your code.*
3. The size (in pixels) to cut from the source image and paste on the target image.
4. The number of pyramid levels to use in the smoothing process.

If a user inputs an invalid file name or size value, your program should handle those errors appropriately.

Convert Images to NumPy Arrays

Use matplotlib to read in the image data as arrays. Be sure that the data type for the arrays is *uint8 integers*, which are integer values that range from 0 to 255. Using an incorrect data type could cause errors as you manipulate the arrays.

Convert Images to Grayscale

When reading in a color image with matplotlib the image gets stored as a three-dimensional array since the red, green, and blue values are all stored in their own matrix in the array. By converting the image to grayscale, you will only have to work with a two-dimensional array where each pixel only has a value for how bright that pixel should be.

There are several formulas to convert an image to grayscale. One method is to take the red, green, and blue arrays from the color image and separate them into their own arrays. Then, scale the values in each of those arrays by a constant value and add them all together. As a formula, it would look like this:

$$M_{gray} = 0.289R + 0.5870G + 0.1140B$$

Where M_{gray} represents the grayscale array, R is the array of red values, G is the array of green values, and B is the array of blue values.

When displaying grayscale images using matplotlib's `imshow` function, be sure to use the correct arguments for displaying a grayscale image.

Get the Source and Target Locations

Using the function `get_click_coordinates` in the provided function file `get_location_in_image.py,` you will get the x and y pixel coordinates for where to cut from the source image and where to paste in the target image from mouse click inputs from the user. You will also need to use these locations to create the image masks.

The `get_coordinates_in_image.py` file has two functions in it:

- `onclick(event)`
    - This function stores event information from user interactions in a matplotlib window. In this case, when a user clicks on an image displayed by matplotlib, the x- and y-values associated with the location of the click get stored.
    - *Note: Your program will not call this function.*
- `get_click_coordinates(gray_image)`
    - This function takes a grayscale image as an input.
    - It then displays the grayscale image and waits for a user to click on the image.
    - After the user clicks on the image the, the integer value of the x- and y-coordinates closest to the click location are stored, and the user interface closes.
    - This function returns `(x, y)`
        - This is a tuple containing the x and y pixel coordinates as integers.

Applying a Gaussian Filter

Image filters are a type of convolution. Applying a Gaussian filter to an image creates a smoothing effect that results a blurred image. For this project use this 5x5 Gaussian filter:

$$g = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

This convolution is given by the double summation

$$F(i,j) = \sum_{k=0}^{4}\left(\sum_{l=0}^{4} g(k,l) \cdot I(i+k, j+l)\right)$$

Where

F is a matrix representing the resulting, filtered image

I is a matrix representing the original image

g is a matrix representing the Gaussian filter

i, j are the indices of the of the images

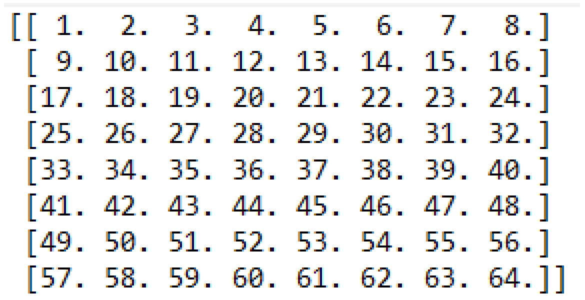
k, l are the indices of the Gaussian filter

Your filtered image needs to be the same size as the original image after this process, so your team will need to decide how to handle the edge cases for this process. For instance, when the filter is centered on (0, 0) the first two rows and columns of the filter do not overlap the image. Zero padding or resizing the filter are two options for resolving this issue.

An image of a penguin before (left) and after (right) a Gaussian filter. Zoom in to see the difference more clearly.

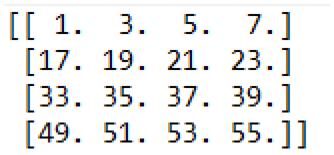
Subsampling

After applying the Gaussian filter, your images will be subsampled. One method of subsampling is to remove every other row and column of pixels from the image. For example, if you had an 8x8 array, it would get subsampled to a 4x4 array.

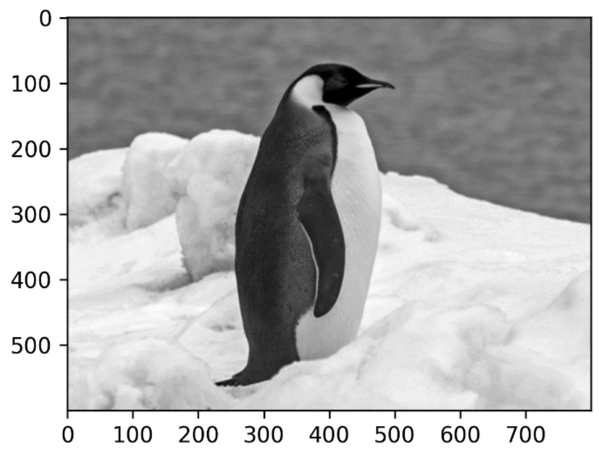


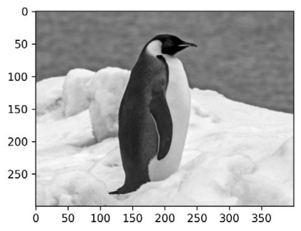
```
[[ 1.  2.  3.  4.  5.  6.  7.  8.]
 [ 9. 10. 11. 12. 13. 14. 15. 16.]
 [17. 18. 19. 20. 21. 22. 23. 24.]
 [25. 26. 27. 28. 29. 30. 31. 32.]
 [33. 34. 35. 36. 37. 38. 39. 40.]
 [41. 42. 43. 44. 45. 46. 47. 48.]
 [49. 50. 51. 52. 53. 54. 55. 56.]
 [57. 58. 59. 60. 61. 62. 63. 64.]]
```

$\longrightarrow$



```
[[ 1.  3.  5.  7.]
 [17. 19. 21. 23.]
 [33. 35. 37. 39.]
 [49. 51. 53. 55.]]
```





An image of a penguin before (left) and after (right) being subsampled. Notice the change in the numbers on the axes.

Upsampling

Upsampling an image is essentially the opposite of subsampling. To upsample, rows and columns are inserted between each existing column and row. The question is, what values should the pixels in the inserted rows and columns have? To determine those values your team will need to implement an algorithm to interpolate the data that goes in the newly added rows and columns. In this project we will use two different methods to find these values.

Take the 3x3 grid below as an example. Each cell represents the value of a pixel in an image and the gray cells are ones that have been inserted during upsampling whose values need calculated. A method called bilinear interpolation allows us to calculate the central value of this grid using the four known values in a series of calculations.

5	▨	10
▨	▨	▨
1	▨	19

First, calculate the averages along the x-axis. *(Note: These averages are NOT the values that go in the top and bottom gray cells.)*

$$x_1 = \frac{5+10}{2} = 7.5$$

$$x_2 = \frac{1+19}{2} = 10$$

Then, take the average of those two values to get the central pixel value.

$$avg = \frac{x_1 + x_2}{2} = \frac{7.5+10}{2} = 8.75$$

5	▨	10
▨	8.75	▨
1	▨	19

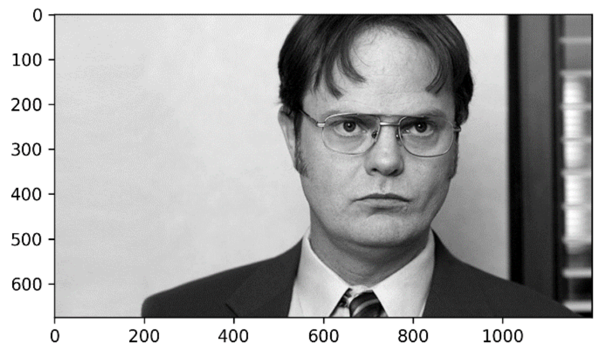
This calculation will need to be repeated for each unknown pixel value that has known values in its diagonally adjacent neighbors. *(Hint: Is there a pattern to the location of these values that you can use to your advantage?)*

For the other missing values, you can use a similar method of averaging, except that instead of using averages of the diagonal neighbors, you can use the average of the cells directly above, below, left, and right of the value you are trying to calculate.

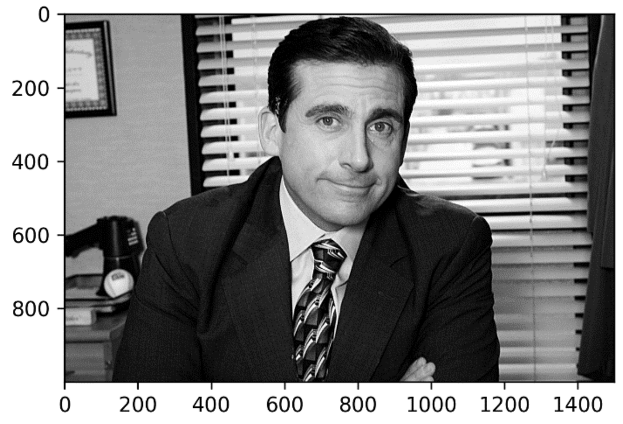
Creating an Image Mask

Image masks are used in a variety of image processing applications. In this project they will be used to tell the program where the source and target images should remain the same and where they should be blended. The image mask itself is an array that your program will create that is the same size and shape as the image it will be applied to. When the mask array is created, the array will only contain ones and zeros. To apply an image mask, you multiply the mask and image arrays by taking each pixel value in an image and multiplying it by its corresponding value in the mask. So, anywhere there is a one in the mask, that pixel value will be preserved. Anywhere there is a zero in the mask, that pixel value will become zero and would then appear as black.

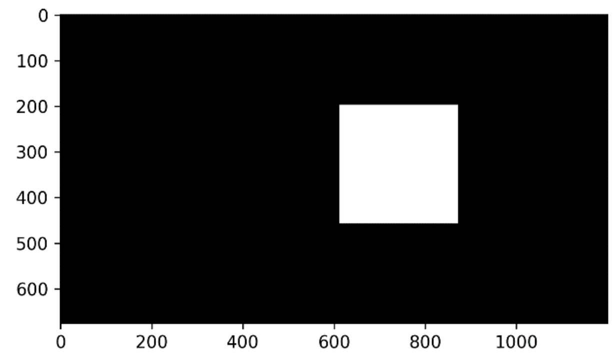
Here is an example where Dwight Schrute (left) is the source image and Michael Scott (right) is the target image. To cut out Dwight's face and place it on Michael's we would need masks that look like the ones below. Note that the squares are the same size (in number of pixels) so that the "cut" and "paste" regions will match. In the numerical arrays, black has a value of 0 and white has a value of 1. (The image at the top of this document is the final product of blending these two images.)



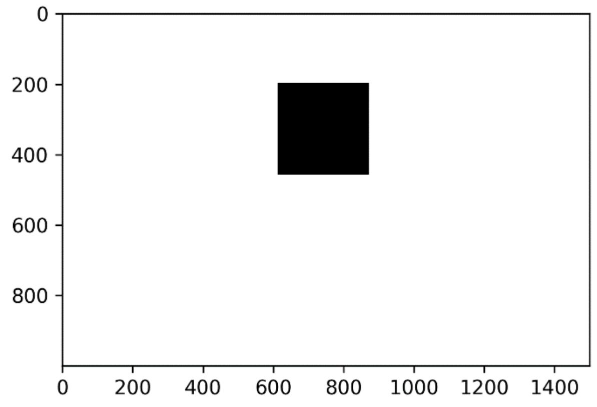
The source image.



The target image.



The source mask.



The target mask.

Creating a Laplacian Image

A Laplacian image is the result of the following process.

1. Get an image
2. Blur the image using a Gaussian filter
3. Subsample the blurred image
4. Upsample the image from step 3
5. Blur the image from step 4 using a Gaussian filter

6. Subtract the resulting image array in step 5 from the image array in step 1

This process leaves you with an image that contains only the details and sharp edges from the image you started with (see the figure below). One of these images will be created for each level of the pyramid.

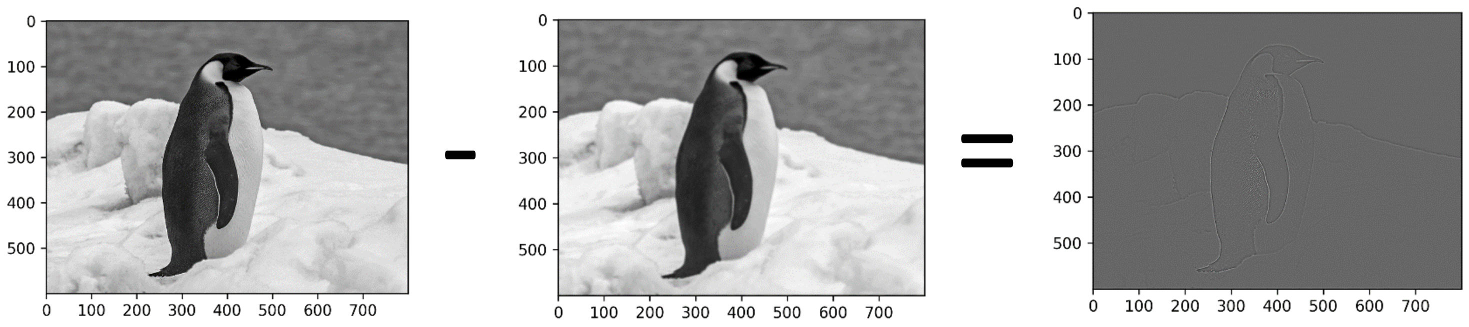


Image from step 1 *Image from step 5* *Resulting Laplacian Image*

Deliverables

Each team will submit a project report outlining their research into image processing, the applications of image processing in a particular engineering discipline, and the program the team developed. The project report will follow the common structure of a technical report. The Python files, along with a report, will be uploaded to Gradescope as the final submission. Make sure that the files will run as they are uploaded (e.g. have them all in the same folder). You may zip all your files together. The report will consist of:

1. **Project Motivation**

   You will need to describe how image processing methods are used in an engineering discipline citing at least three credible sources using either APA or MLA format. This section will outline the major problems trying to be solved in these areas and the state-of-the-art methods that have been developed or are widely used.

2. **Project Overview and Methods**

   In this section, you will describe the background of image pyramids and how they are used for blending images. You will need to summarize the theoretical background of the technique used in this project.

3. **Discussion of Algorithm Design**

   This section will include flow charts of the overall program and each of the user-defined functions. This section should describe the design rationale for modularizing the code to meet the needs of future uses of image blending. How will future users be able to use your program to process their own images? You will need to summarize the algorithm you designed and used for Python programming. Answer the questions posed in the instructions in this part of the report.

4. **References:**

List of resources used to research the applications of image processing in different engineering disciplines and any additional resources referenced that your team used to help write your code.

5. **Appendices: Include as appendices to your main report**
   a. **User manual** with sample inputs and outputs (including pictures).
   b. **Project management Plan**: Summary of the contributions of each member of the team to the project. Describe your team's methods for collaboration that facilitated active participation and learning by all team members. Include a discussion of opportunities for improvement for future team projects.
   c. **Discussion of Design Process**: Description of how your team followed the design process, which is defined as "a process of devising a system, component, or process, to meet desired needs within constraints. It is an iterative, creative, decision-making process that involves developing requirements, performing analysis and synthesis, generating multiple solutions, evaluating solutions against requirements, considering risks, and making trade-offs."
   d. **Code:** The team should upload the Python files, but they should also include the code as an appendix in the report. Code should be well commented to increase readability for future users of the algorithm.

Project Grading Summary (a detailed grading rubric will be provided):

Points	**Code**
70	Code runs with no errors, and image processing and template matching portions are implemented using only `matplotlib` and `NumPy` modules
6	Errors handled and structured appropriately
10	Comments in code, appropriate variable names, code is readable
	Report
5	Project motivation and application to major or career interest(s)
10	Project overview and methods discussion
10	Algorithm Design rationale and Flow charts
5	User manual for code
3	Project Management Plan (Summary of contributions)
3	Discussion of Design Process
3	Appendices include the code
125	**Total**

In class demo

You will need to show the followings:

- Run your program with images provided by the teaching team.
- Run the program for images that your team chooses

- Demonstrate how your program handles invalid inputs and edge cases.