ECE 558 - Project 2
Alex Carter
agcarte4

Section 1: Implementation

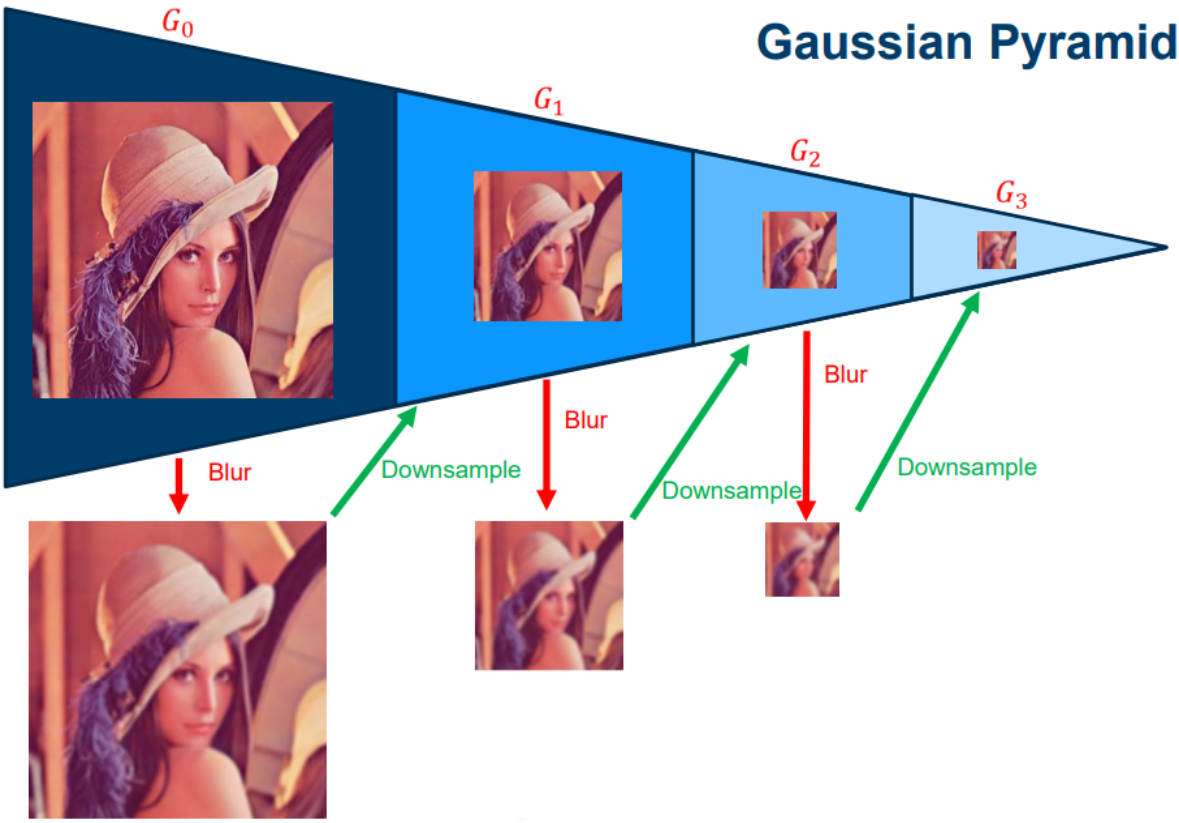
1. Pyramid Computations

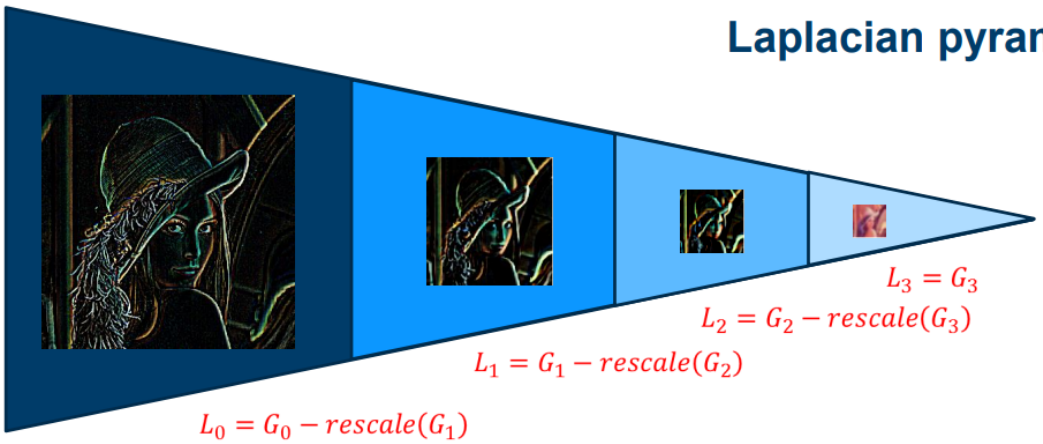
Fundamentally the Gaussian and Laplacian pyramids are simple to compute, once you have the Gaussian then the Laplacian requires only a few more steps. To accomplish this, I created a function as described in the assignment. This function is called *ComputePyr*, it has two inputs and two outputs:

```
gPyr, lPyr = ComputePyr(input_image, num_layers)
```

Providing the function with an input image and a value for the number of layers in the pyramids, it will compute the Gaussian and Laplacian pyramids. The function utilizes code from the previous project, namely the *conv2* function that was implemented. It defaults to using a 5x5 Gaussian blurring kernel with reflection padding. The Gaussian pyramid is computed by blurring then downsampling the base image. This process is repeated for each layer up to the max or the value provided. The Laplacian is computed by taking the difference between the layers of the Gaussian Pyramid.

To call this function (please see the provided python source file), it is necessary to pass in an image and a value for the number of layers in the pyramids (alternatively the function will compute the max). It will return two NumPy arrays, one for the Gaussian pyramid and the second for the Laplacian pyramid. The outputs are NumPy arrays that can be indexed based on the layer that you one wants to access.





Graphical representation of Gaussian and Laplacian pyramids
(Borrowed from these lecture notes by Idar Dyrdal)

2. GUI

The graphical user interface (GUI) was implemented using Python's native Tkinter library. Some of the source code for this implementation was repurposed from my senior design project; this was done to help me get started with its development. This is organized as a class and the GUI starts when the user creates a GUI object. The constructor for this class takes a Tkinter object, a window name, and a value for the number of layers in the pyramids.

Within the GUI you will find a file menu and several buttons at the bottom of the window. Under the file menu, you can import a foreground and background image for use with Laplacian pyramid blending. In addition there are also options to save the blended image, clear the image canvas, and exit the GUI. Please note that the user **MUST** import a foreground image before importing the background image. On the bottom of the GUI is a row of buttons, each with their own command. There are three buttons to compute the pyramids for the various images (the foreground, background, and mask) that can be loaded into the GUI. There are two buttons for creating masks: a button to create a rectangular mask and one to create an elliptical mask. One button will initiate the Laplacian blending process. The last button will allow the user to save the resulting image from the blending process.

The images are displayed using a Tkinter canvas object, with the left image being the foreground and the right image being the background. Once the blended image has been computed, it will show up on the bottom.

Please note that the GUI is not robust, there are some bugs that will cause it to lose functionality. I didn't put too much time in debugging and accounting for user errors. It is probably best to simply restart it if something stops working. Also, the computations using convolution are slow, please give these time to complete.

An example on how to start the GUI can be found in the source code, near the bottom of the file. But it is as follows:

```
master = GUI(tk.Tk(), "Project 2")
```

This creates the main GUI window and starts the Tkinter window loop. Once it has started, the user can exit the program with the close button, exit from the file menu, or force-stop the script from executing.

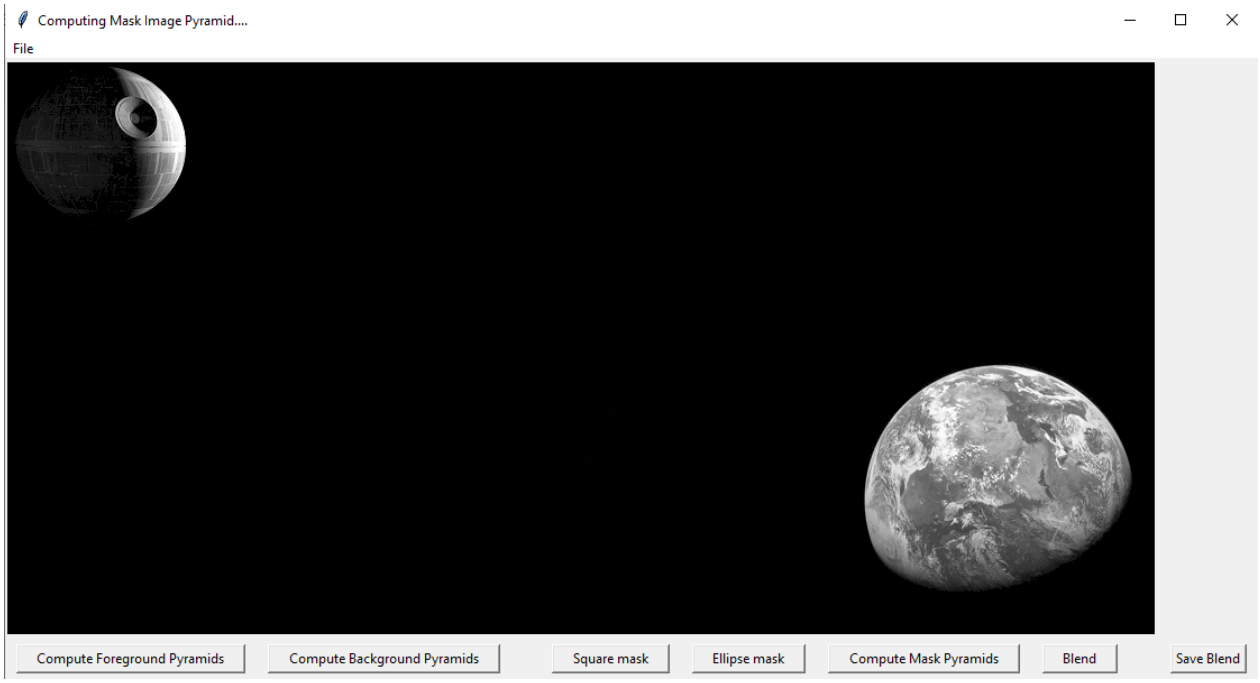
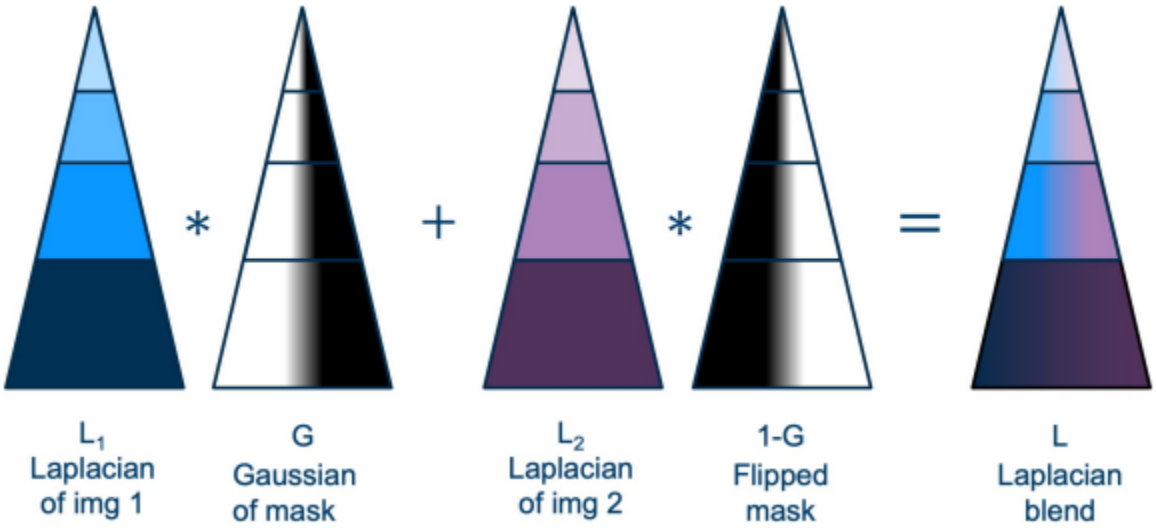


Image of the base GUI

3. Laplacian Blending

The algorithm to compute the blending of the Laplacian pyramid is fairly simple. For each layer in the pyramids, compute the blended Laplacian pyramid as shown below. Apply the mask to the foreground image, which isolates the region that the user wants blended onto the background image. The inverse of the mask is applied to the background to remove the region that the foreground will take. Once the blended Laplacian pyramid has been computed, collapse the pyramid to the final blended image. Collapsing the image involves starting from the top layer, upscaling it, applying Gaussian blurring and then adding it to the layer below. This is repeated until all layers have been upscaled, blurred, and added to one another. Ideally, the final result will be a perfect recreation of the original images but with the mask applied.



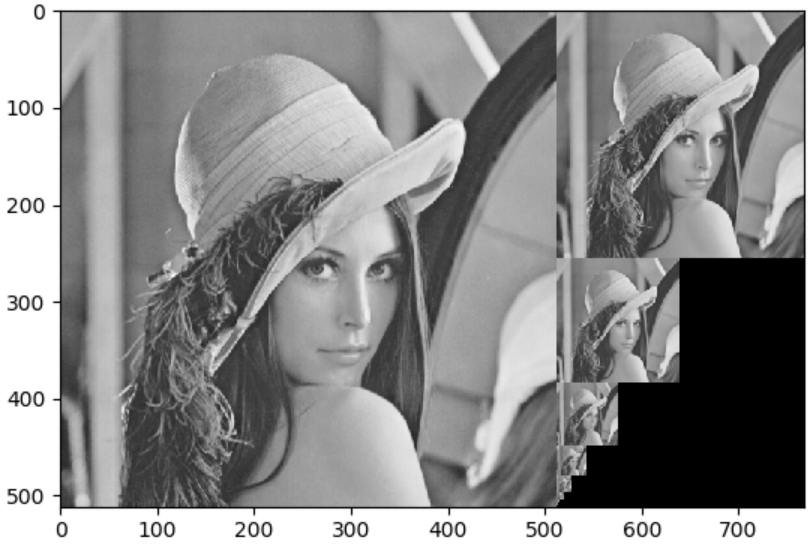
ECE 558 - Project 2
Alex Carter
agcarte4

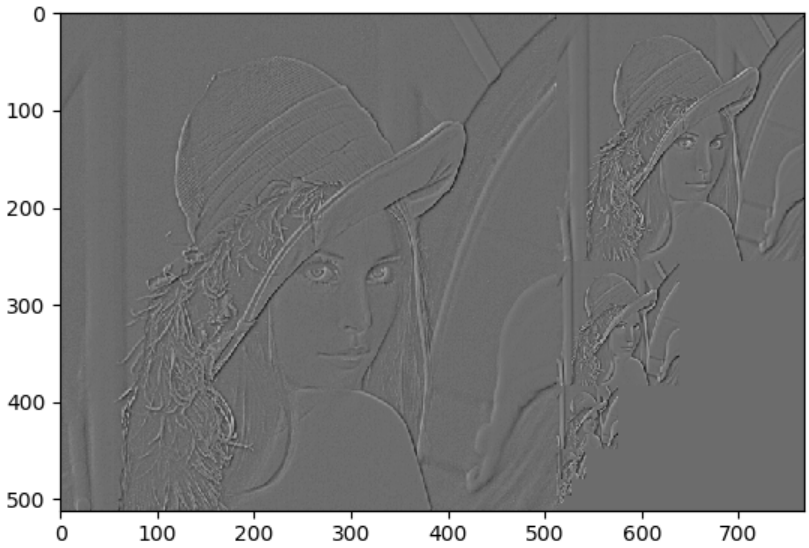
Single layer of Laplacian blending

Section 2: Results

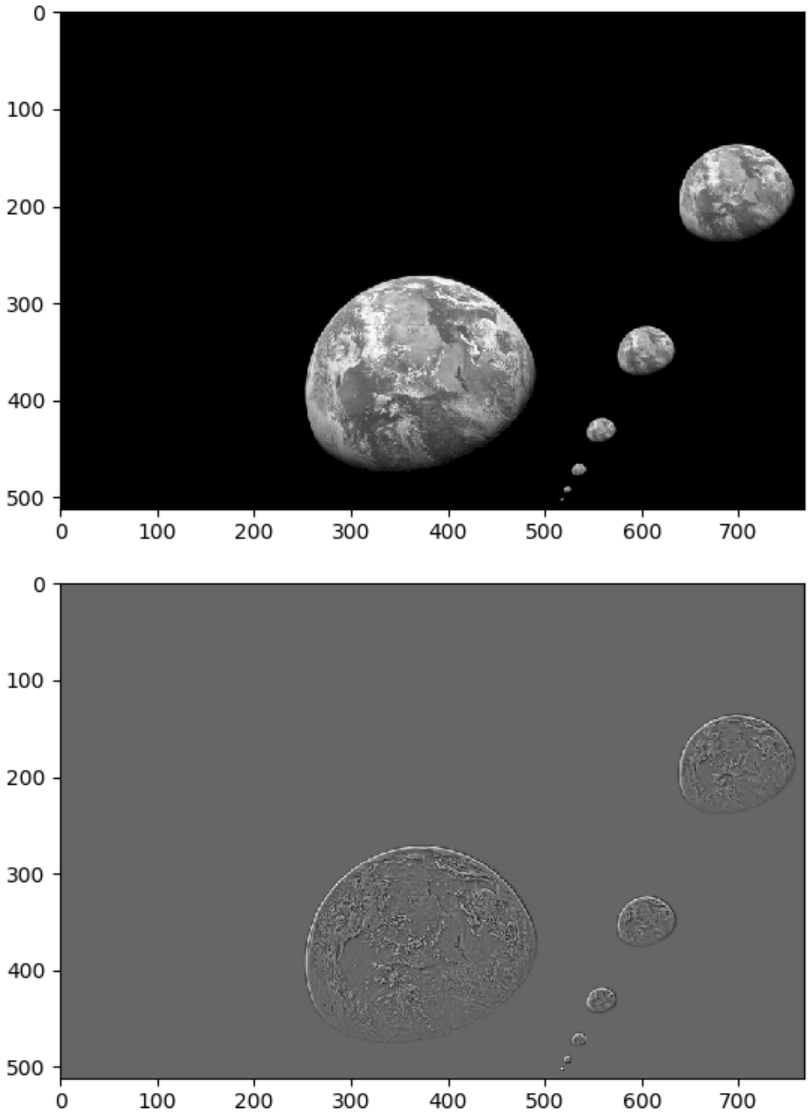
1. Pyramid Computations

I added in a function to combine the pyramids together so that they would be easy to view. Below are some of my favorites.

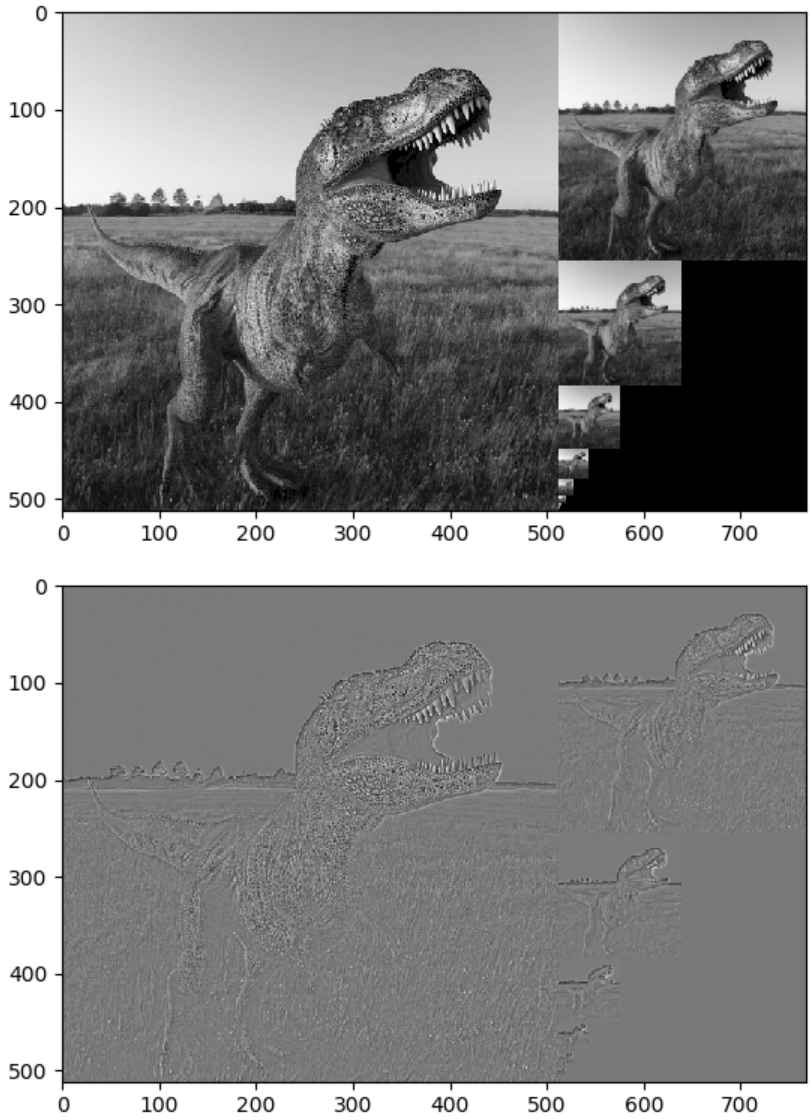




Lena


0
100
200
300
400
500
0
100
200
300
400
500
600
700
0
100
200
300
400
500
0
100
200
300
400
500
600
700

Earth

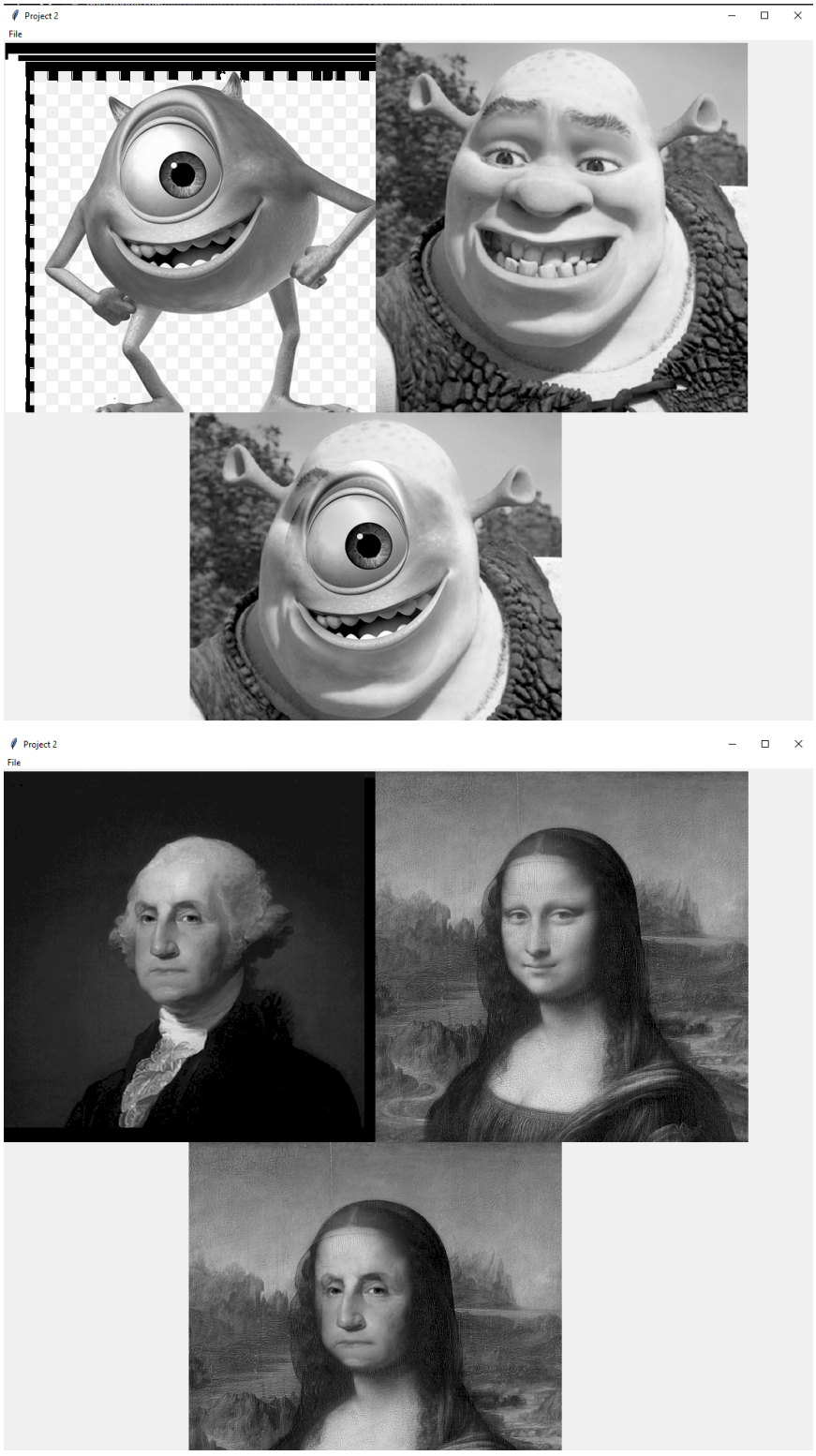


T-Rex

This function was extremely useful, it helped me debug when something wasn't working as intended. It also is nice having visuals of the pyramids when needed. This also helped me cross-check my work with other resources, as I could verify that these pyramids were being correctly computed by my own algorithm implementations.

2. GUI

The repurposed GUI code worked extremely well for what it is,it will get all of the basic jobs done. There isn't much to say about it other than mentioning that there are bugs and issues that can cause it to not function properly. However, I feel that this is acceptable given that this component isn't the main focus of the project. I was lucky enough to have had experience with Tkinter in Python.



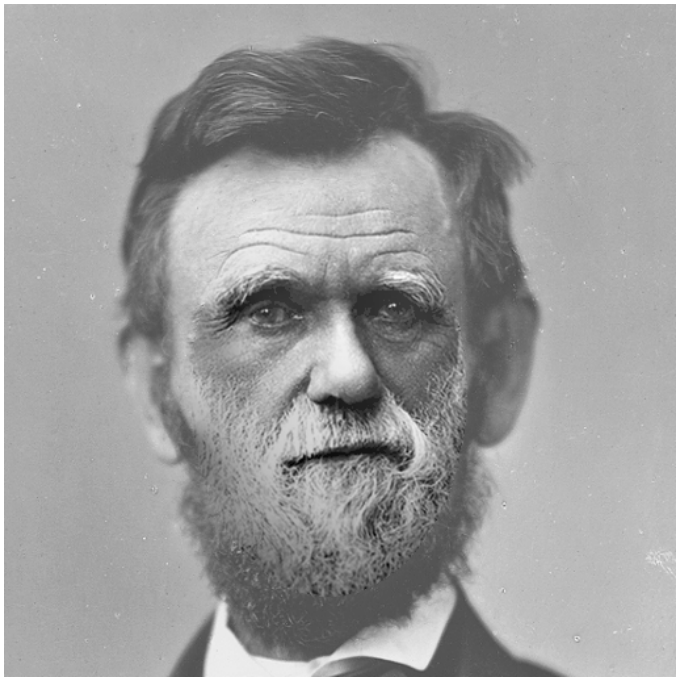
GUI after blending

3. Laplacian Blending using GUI

I am pleased with how well the GUI works in conjunction with the pyramid blending. I will say that I spent much time figuring out the specifics with Laplacian blending. At first my blended images looked like the originals, but the result seemed as if it was aliasing in some way. After some time debugging I realized that I needed to perform Gaussian blurring when I was performing the upsampling during the pyramid collapse.

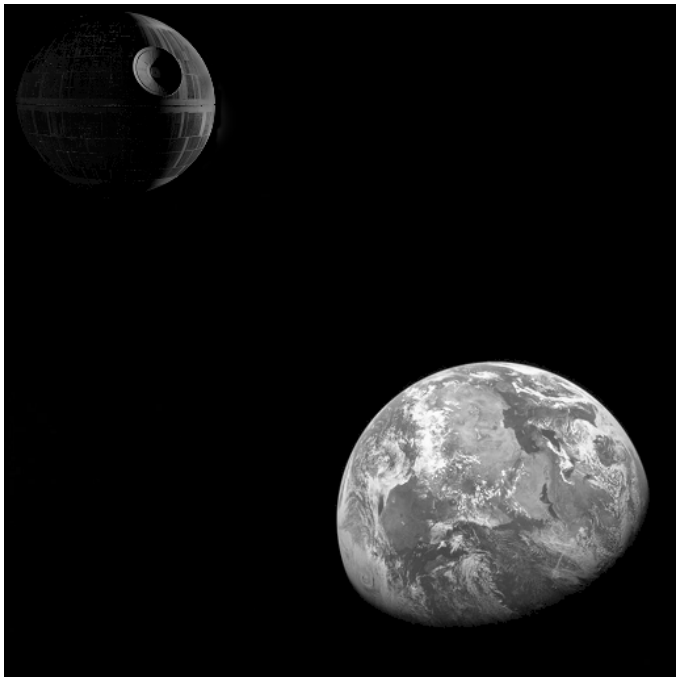
Below are a few of my favorite blends:

Loch Ness Monster in Reflecting Pool



ECE 558 - Project 2
Alex Carter
agcarte4

Charles Darwin on Abe Lincoln's Portrait



Death Star approaching Earth

Mike Wazowski as Shrek

As seen in the above results, the Laplacian blend produced great results. There are a few obvious mis-matched portions in the images, but these could be avoided by better aligning the images beforehand with a photo editing tool.

ECE 558 - Project 2
Alex Carter
agcarte4

Section 3: Areas for Improvement

I would like to finish this report by discussing some areas of improvement that I have thought about now that I have come to an end of this project. There are 5 main items that I believe are the most important:

- RGB support
- Improve the conv2 function
- More robust GUI
- Free-form mask selection
- Allow user to select mask location on background image

RGB Support

This is the most important item that could be improved upon. I should have implemented this as I was working through, but due time constraints and other obligations I simply did not want to spend extra resources on this item. Looking back, it is fairly simple to implement, only needing to split the RGB images into their channels to perform the pyramid blending and computations.

conv2 Improvements

Although unnecessary, I feel that the program as a whole would benefit from making the conv2 function a parallized function. Convolution between matrices is extremely paralizable, but this task is brutal when performed by a single thread. This would offer tremendous speedups in the computation of the pyramids and blending. Also may benefit future work that may need to utilize the conv2 function.

GUI Improvements

The last three I will clump together as they all relate to the GUI. As I stated earlier, the GUI is not robust and can easily be broken if user actions don't follow the correct flow. This is a mundane task, the GUI can be made more robust by adding more checks and flags for when buttons are clicked. The next two both relate to the mask creation:a free form mask tool would allow for more accurate masks while allowing the user to select mask location on the background image would make the blended images much better. I didn't implement the free-form mask creator as I had no clue on how to create such a tool.

Note: I was unsure how to organize the structure of my code, if there are any questions please reach out to me.