CS 194-26: Image Manipulation and Computational Photography, Fall 2022

Project 2: Fun with Filters and Frequencies!

Ethan Gnibus

Overview

In this project, I will manipulate images using filters and frequency to extract useful information and make cool art. I will use filters to blur images, take the partial derivative in x and y of images, and edge detect images. I will use frequencies to sharpen images, make hybrid images, and blend images together!

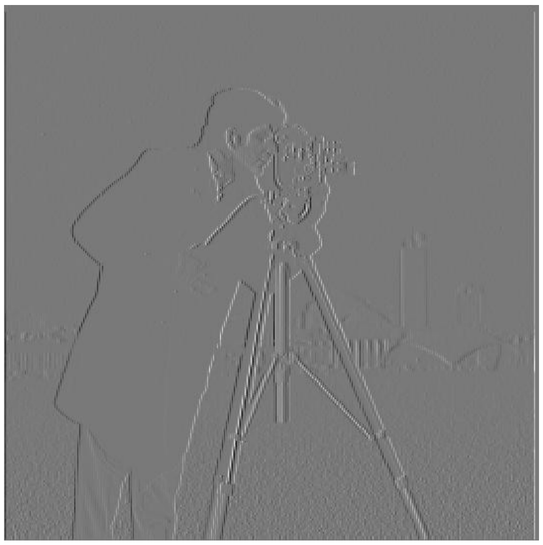
Part 1: Fun with Filters

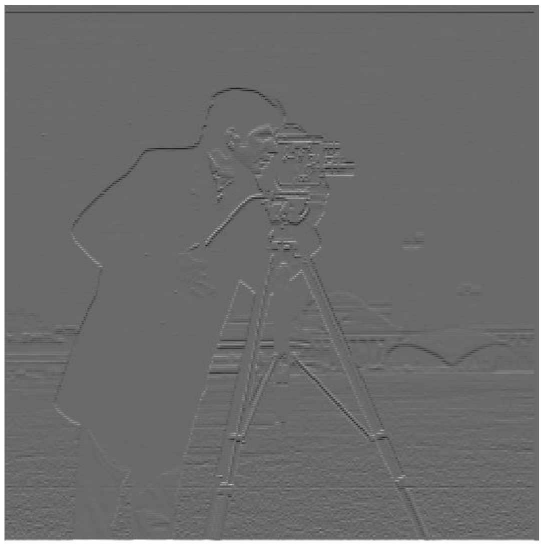
Part 1.1 Finite Difference Operator

Here is the Cameraman Image

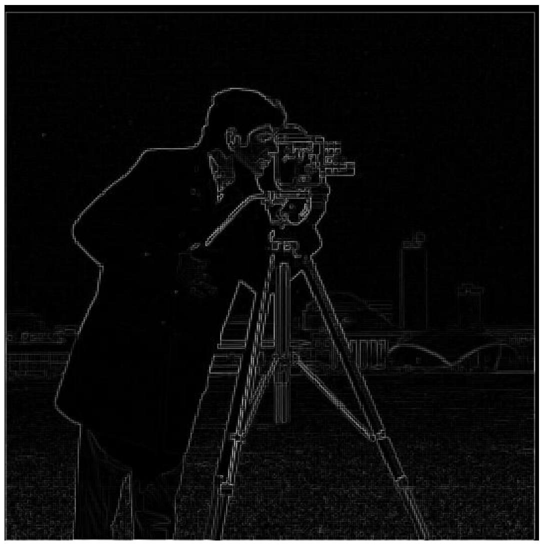
cameraman.png

We can find the partial derivative in x and y of any image using convolution. To do this, we will construct the finite difference operators D_x = [1, -1] and D_y = [1, -1].T. Next we will convolve the original image by D_x and D_y to get the following:

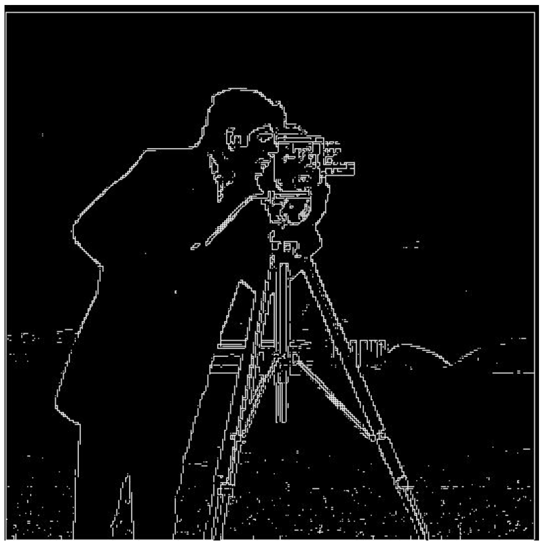

Partial derivative in x


Partial derivative in y

We can combine the partial derivative in x and y to get the gradient magnitude of our original image:


Gradient Magnitude

To edge detect, we can binarize the gradient magnitude by turning pixels white if their value is above a threshold, or turning them black if not:


Bad Edge Detection

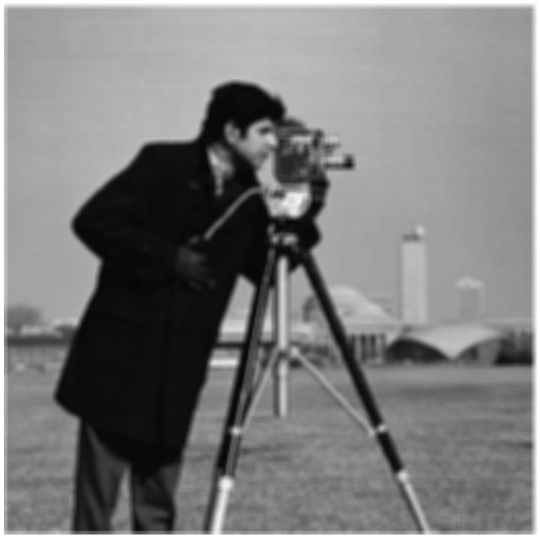
Why Gradient Magnitude works for edge detection:

Using partial derivatives, we are able to find the slope of how fast things change in the x and y directions of our images. If slope is high at a given point, we know that significant change happens at that point. Knowing where significant change in the x and y directions are useful, because we can combine them to see where significant change happenes in both directions! This is essentially what taking the gradient magnitude is. Now that we know where significant change is in all directions, we can use that information to determine how significant that change must be for our algorithm to recognize that pixel as an edge.

Part 1.2 Derivative of Gaussian (DoG) Filter

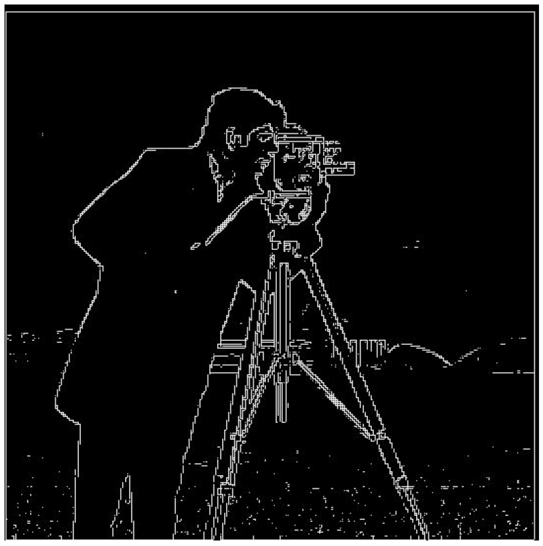
Unfortunately, our edge detection algorithm alone results in lots of unwanted noise. To filter out some of the high frequencies in this image, we will run the original image through a low-pass filter before running our edge detecion algorithm:

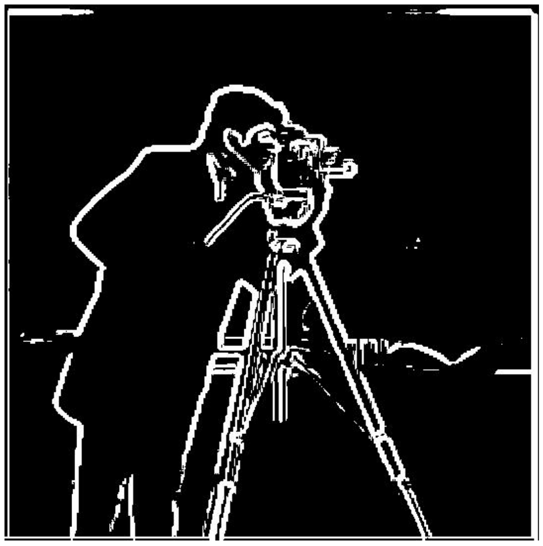
Input Image


Blurred Image

Edge De

Here we can compare our old edge detection algorithm (no blur) and our new edge detection algorithm (with blur)


Edge Detection Before Blur


Edge Detection After Blur

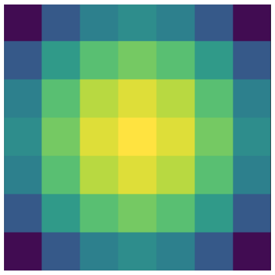
As we can see, when we blur we get much less noise! This is because blurring an image removes it's high frequencies. We can also see that this process has a tradeoff. The edge detection algorithm finds fine lines on the unblurred image, but it finds thick lines on the blurred image. The lack of high frequency data makes our edge detection estimate on lower freqency data, resulting in less drastic change. This shows that there is dynamic between edge detection accuracy and noise reduction that you have to optimize when making an edge detection algorithm.

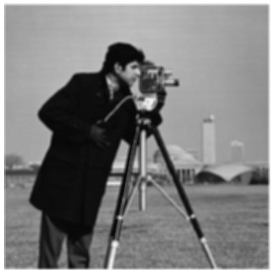
Runtime Optimizations

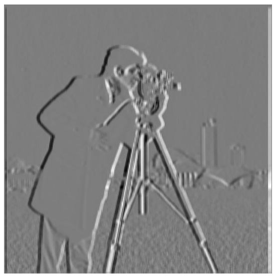
To speed up our algorithm, we can take the derivative of the Gaussian filters, then use them to convolve the input image. This is in contrast to taking the partial derivatives of the original image, then convolving the partial derivatives by a Gaussian filter. Here's a visualization of our two processes:

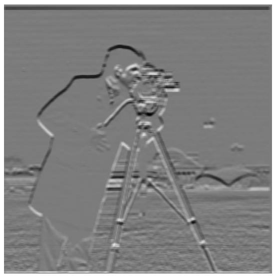
Before Runtime Optimizations

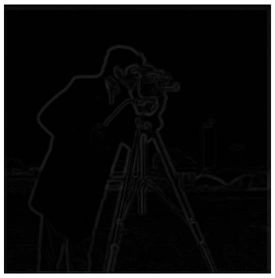
Input Image

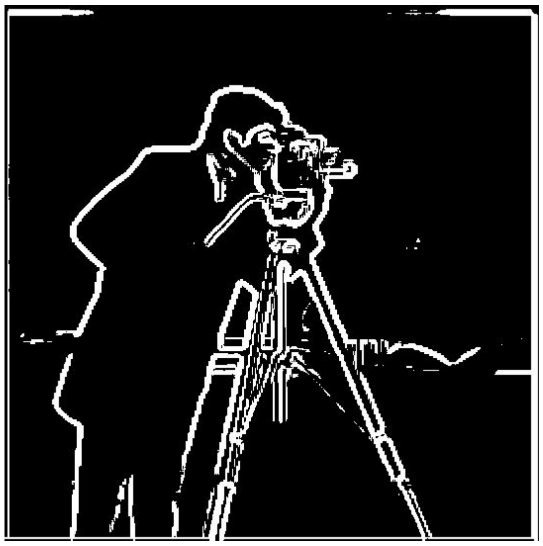

Convolve input image by Gaussian Filter


Blurred Image


Partial derivative in x (one convolution call)


Partial derivative in y (one convolution call)

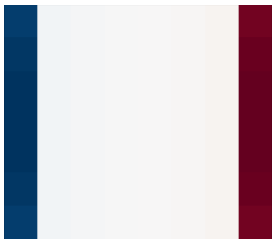

Gradient Magnitude

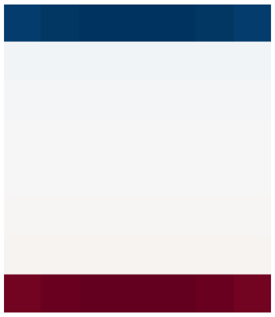

Output Image

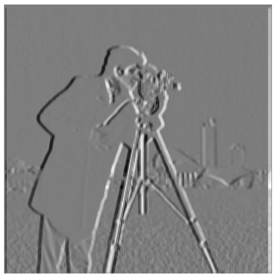
Here we have a total of 3 convolution calls

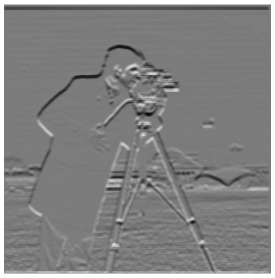
After Runtime Optimizations

Input Image

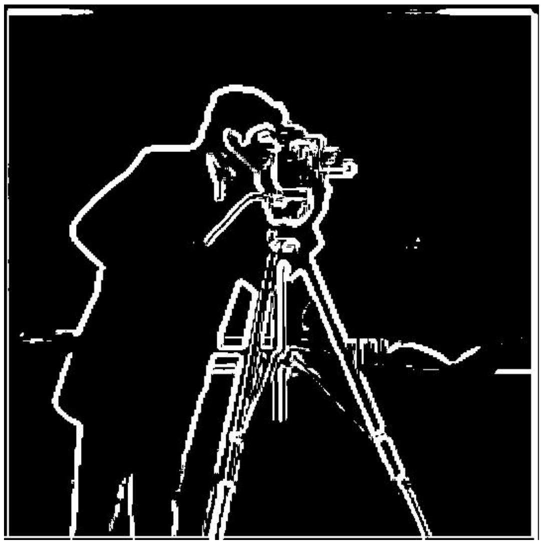

Convolve by Partial Derivative in dx Gaussian Filter


Convolve by Partial Derivative in dy Gaussian Filter


Blurred Partial Derivative in x


Blurred Partial Derivative in y

Gradient Magnitude


Output Image

Here we have a total of 2 convolution calls

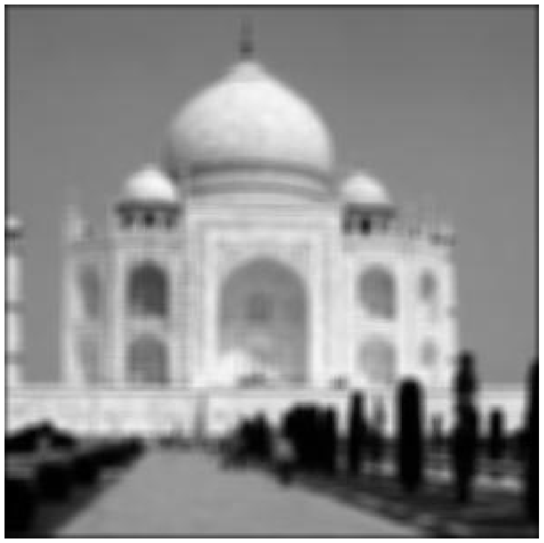
As we can see, we get the same result, but the second approach is much faster!

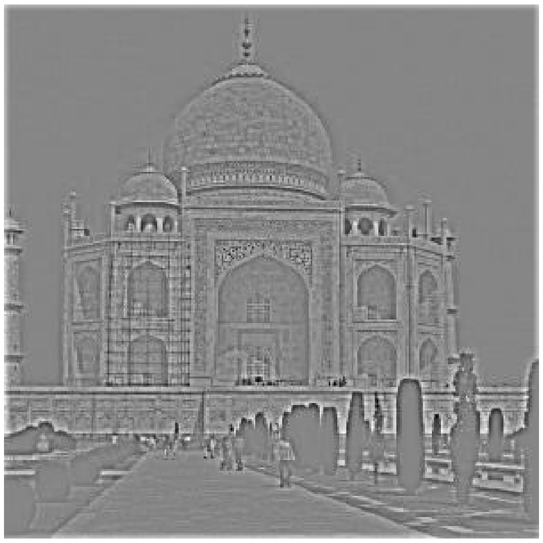
Part 2: Fun with Frequencies!

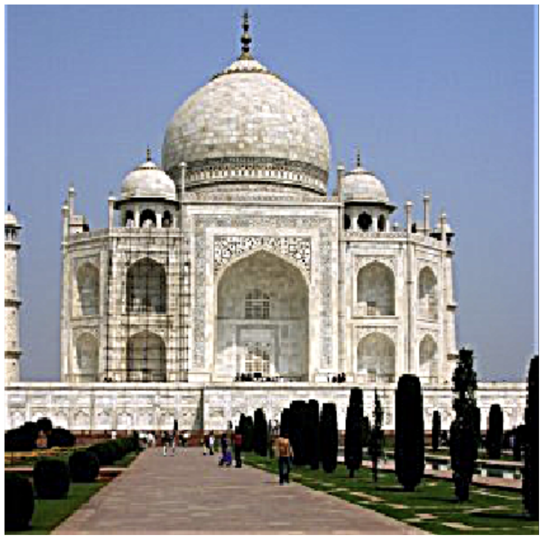
Part 2.1: Image "Sharpening"

Sometimes when you are working with images, you find yourself with a blurry image that you want to sharpen:

Input Image

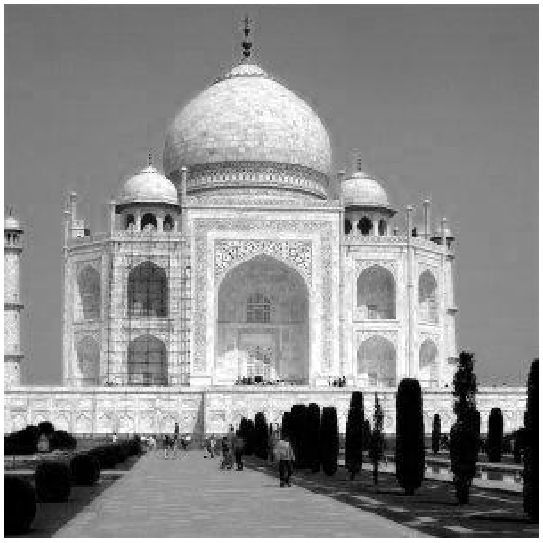

Low Frequencies

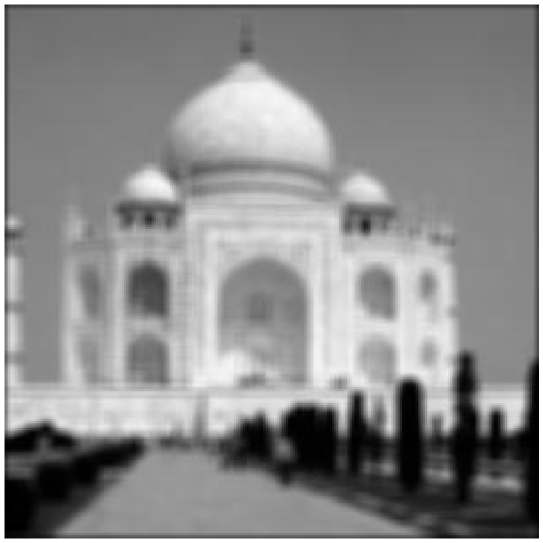

High Frequencies

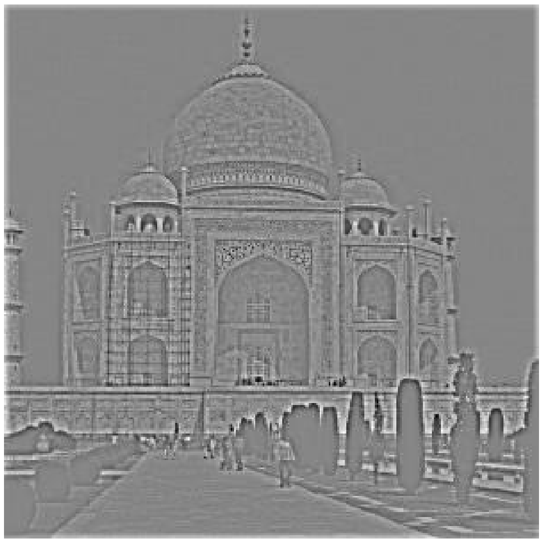

Sharpened image

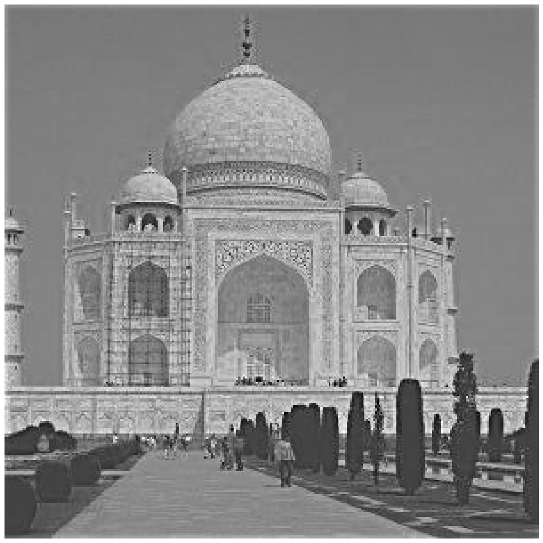
To sharpen an image, we will proceed by making an image's high frequency data more prominent. To do this we will need to find an image's high frequency data, then add it back to the original image. This process entails passing the image through a low-pass filter by convolving it with a gaussian. From this operation we get a blurred version of the original image. Next, we subtract the blurred image from the original image to get the high frequency data from the original image. Finally, we add the high frequency data back into the original image and get a sharpened image!

for simplicity, I will show this process with the just the red channel of an image:


Input Image

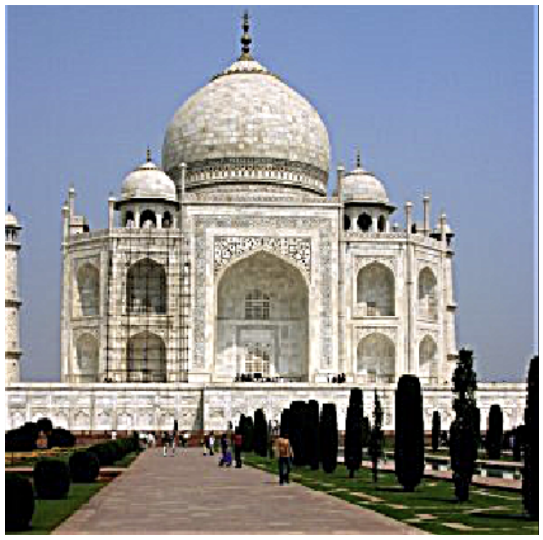

Low Frequencies


High Frequencies


Sharpened image

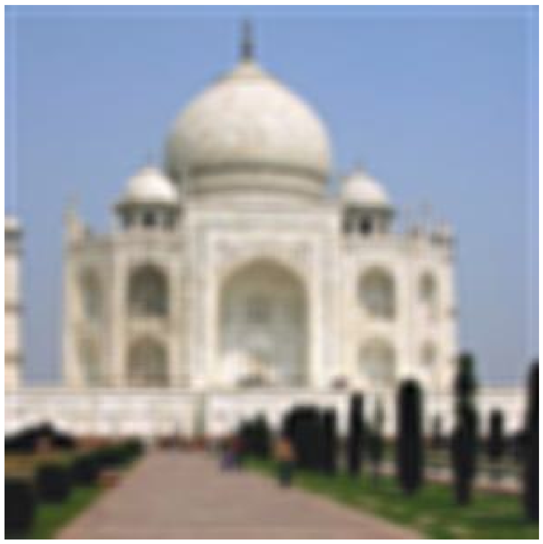
We can combine all these operations into a single convolution operation called the unsharp mask filter. We can construct this unsharp mask filter for any image to sharpen it. This makes sharpening across all color channes really easy:

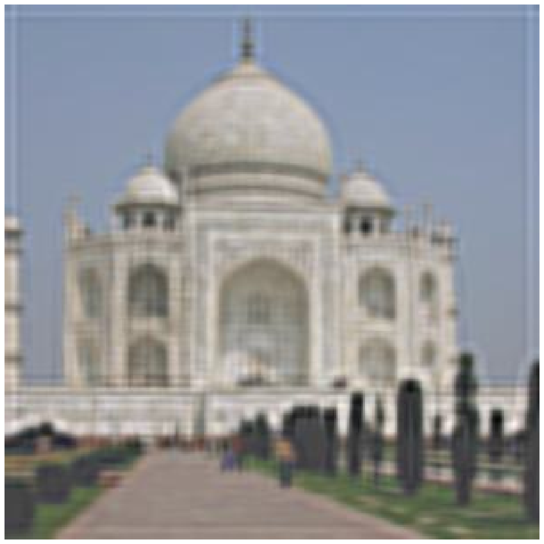
Input Image

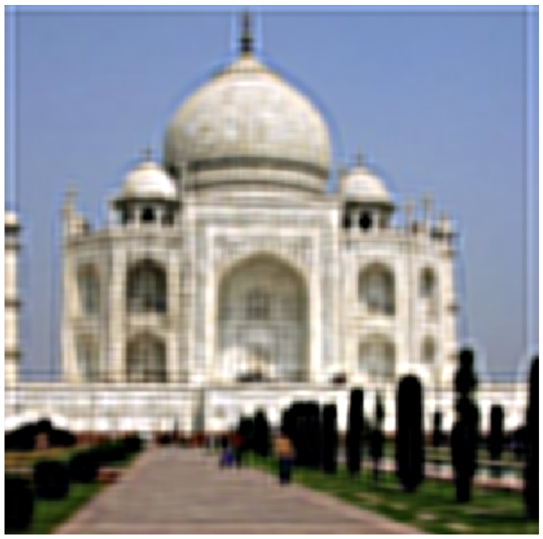

Sharpened Image

Here's what it looks like if we use this unsharp makse filter to take a sharp image, blur it, then try to sharpen it again

Sharp Image


Blurred

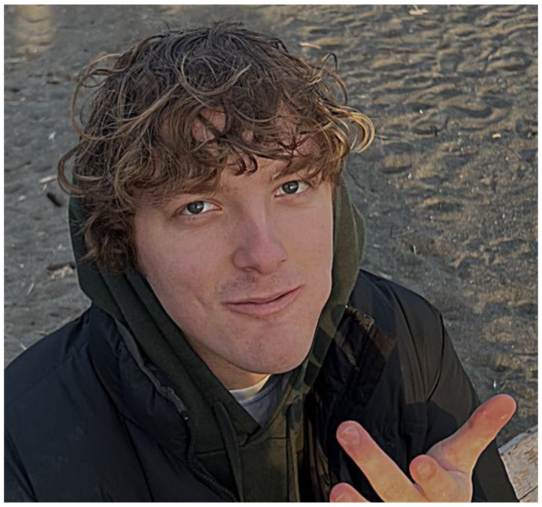

Blurred then sharpened


Blurred then sharpened with small coefficient

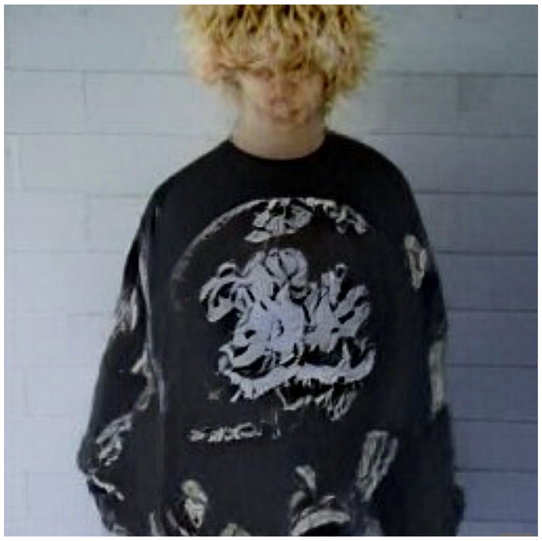
Evidently, resharpening doesn't seem to be as simple as the blurring process. By simply applying the unsharp mask filter to the blurred image, we do not get the original sharp image back. By multiplying the unsharp mask by a small coefficient, we can get a result much closer to the original image.

Unsharpening a few images of my choice

Input Face


Sharpened Face

Input DALL-E Mini

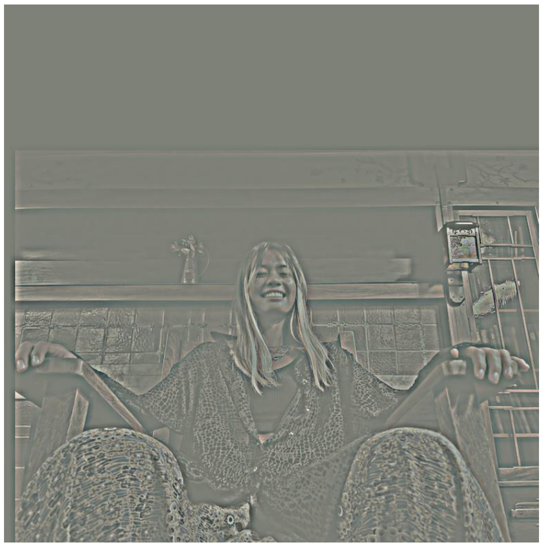

Sharpened DALL-E Mini

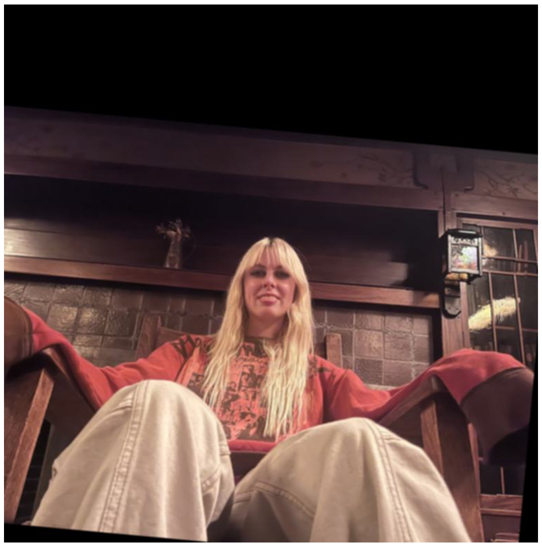
Part 2.2: Hybrid Images

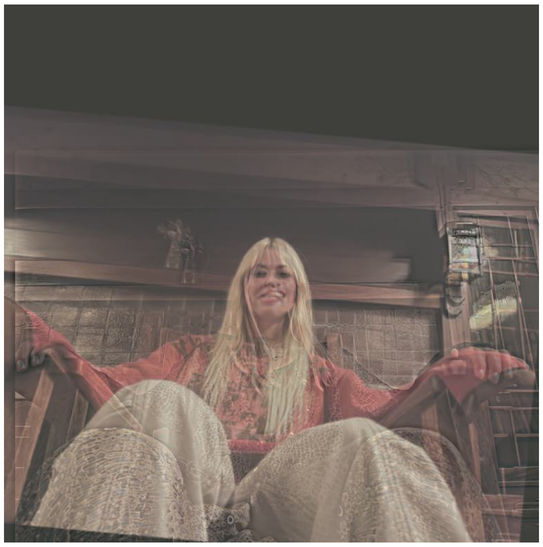
We can combine the low frequencies of one image with the high frequencies of another to make a hybrid image! This hybrid image will look like the first image from far away and the second image from up close. An example is below:

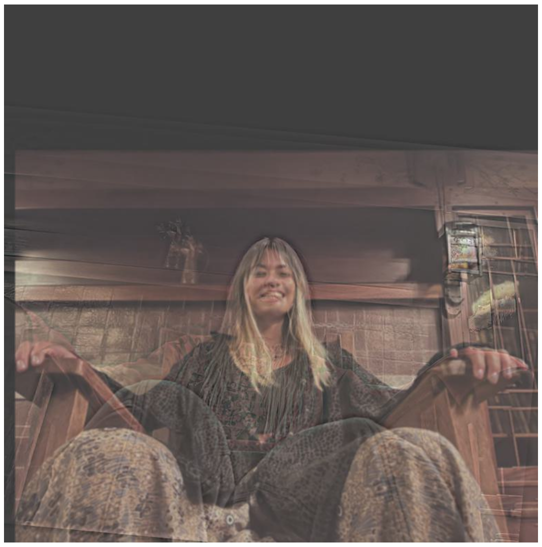
Derek

Nutmeg

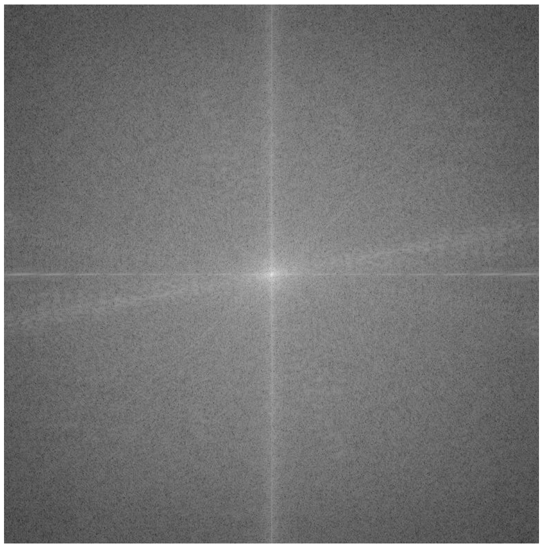

Derek High Frequencies

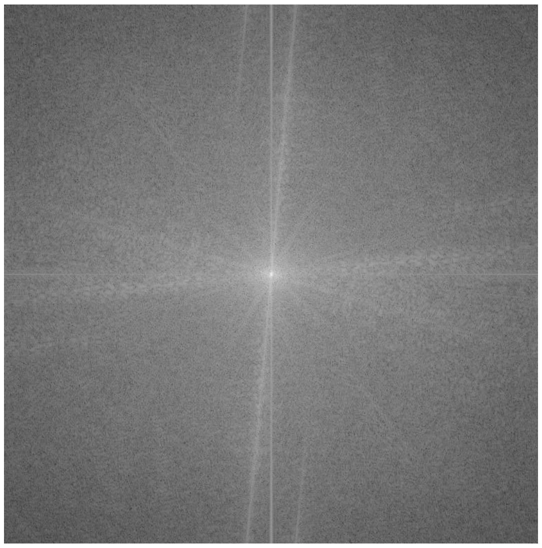

Nutmeg's Low Frequencies

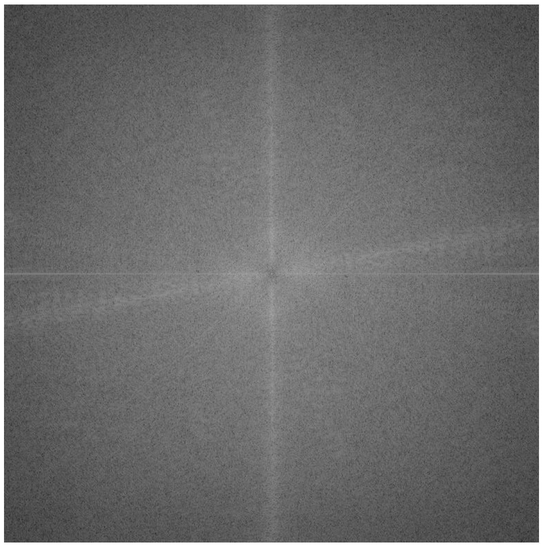

Hybrid Frequency

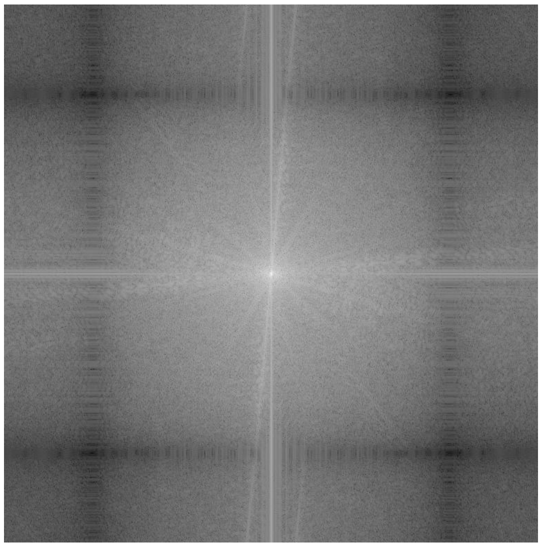

Hybrid Frequency (high's and low's swapped)

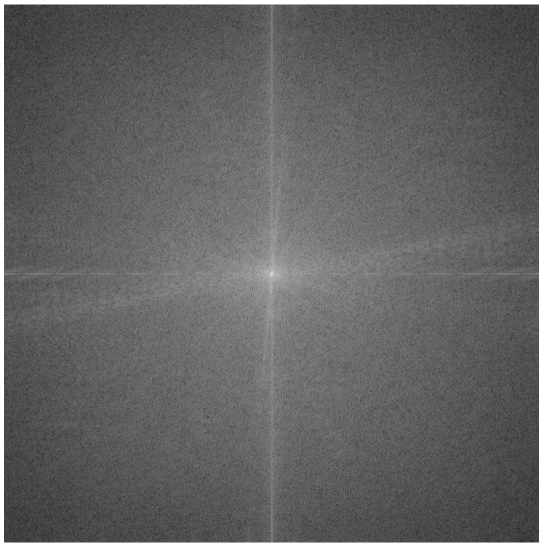
Here we can see the log magnitude of the Fourier transform of the two input images, the filtered images, and the hybrid image.


Derek


Nutmeg


Derek High Frequencies

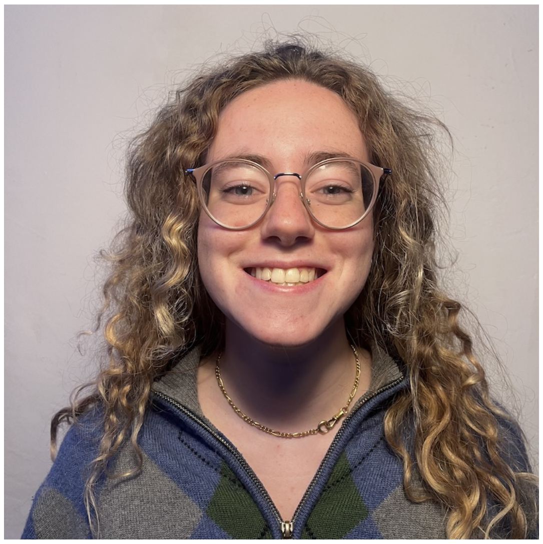

Nutmeg's Low Frequencies

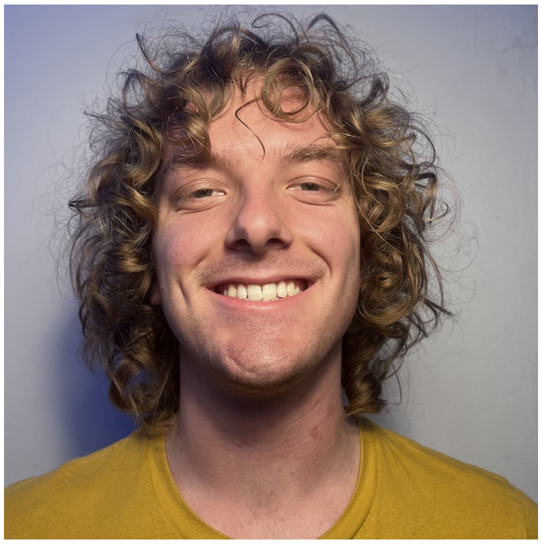

Hybrid Frequency

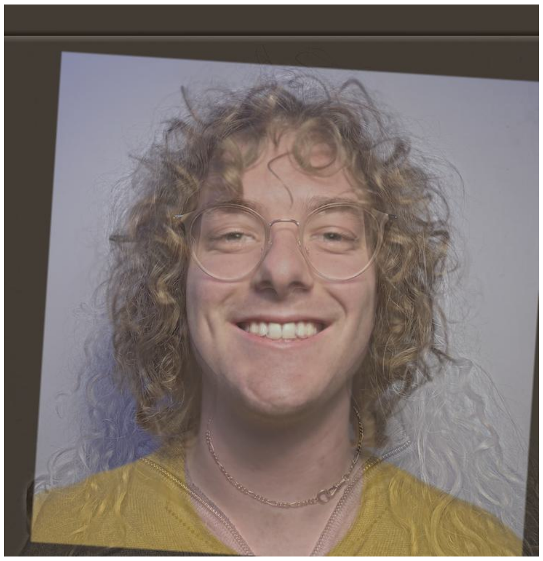
As we can see, the low-pass filter only lets the low frequencies of the first image through. The high-pass filter only lets the high frequencies of the second image through. Consequently, the hybrid image has the low frequencies from the first picture and the high frequencies from the second.

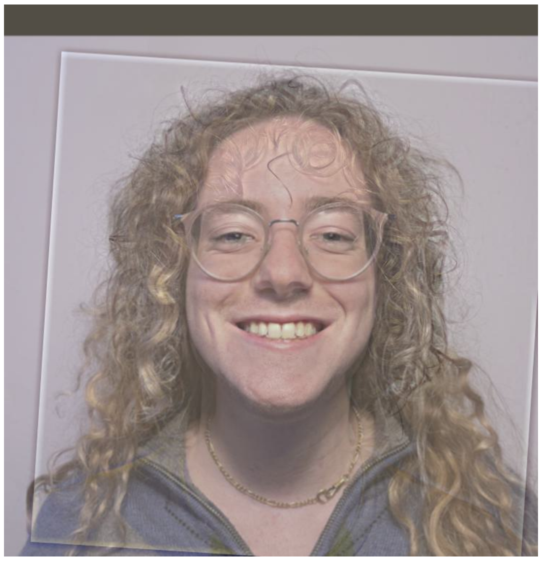
Here I experiment with some hybrid image results:

Mixing Ethan with Juliette


Juliette


Ethan

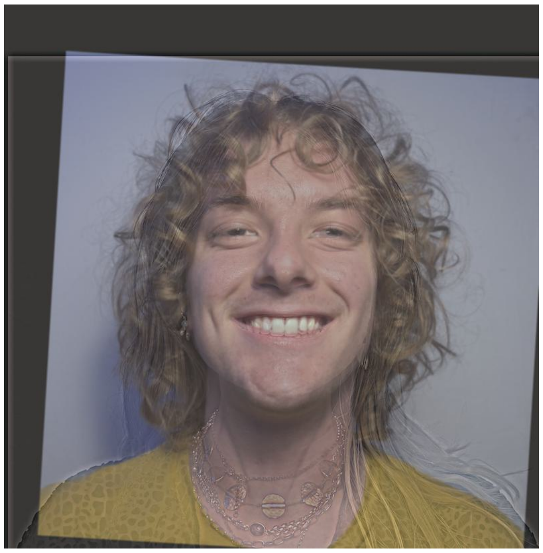

Mixed

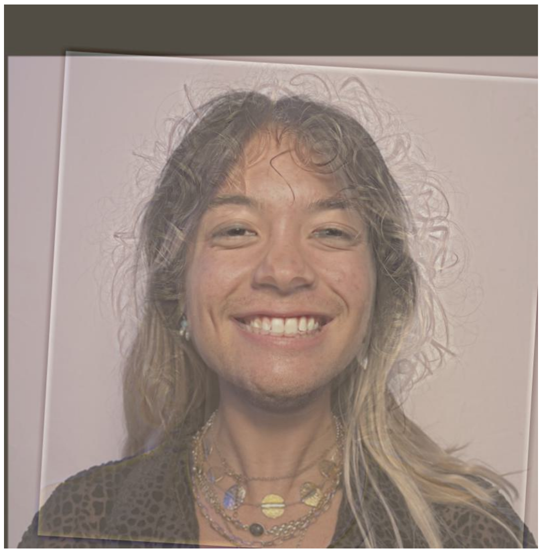

Mixed Flipped

Mixing Ethan with Emma

Emma

Ethan


Mixed


Mixed Flipped

Bells & Whistles

I chose to implement hybrid images with color and found that the images looked the best if I used full color for both the high and low frequencies

Multi-resolution Blending and the Oraple journey

Here we want to make a seamless transition between two pictures by blending across frequencies. To do this we will make Gaussian Stacks and Laplacin Stacks for both images, combine both stacks using a mask, then add all layers of the combined stack to get our resulting blended image.

Part 2.3: Gaussian and Laplacian Stacks

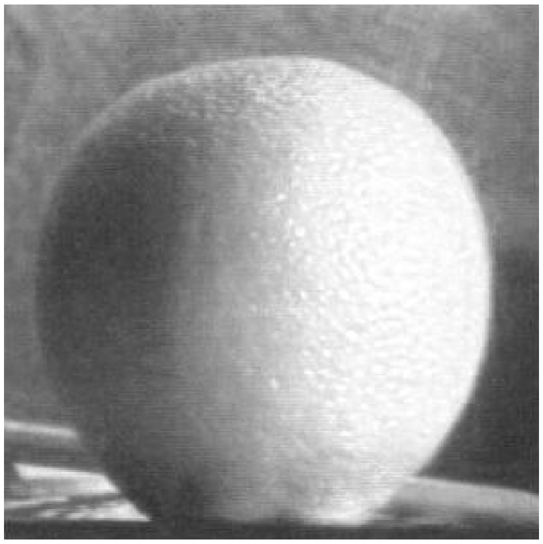
Gaussian Stack

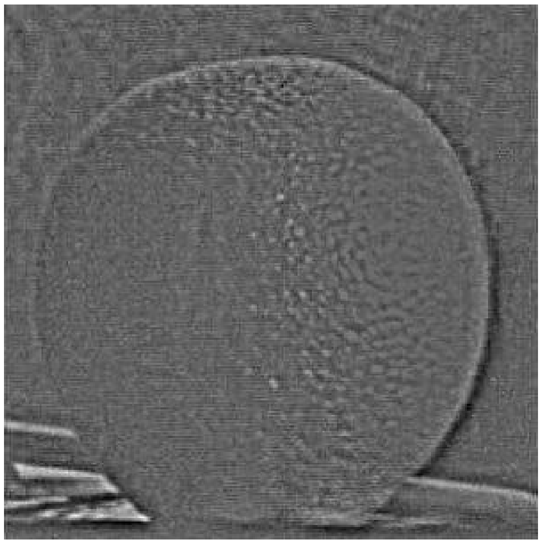
A Gaussian Stack is a collection of same-sized images made by repetitively blurring an input image and saving it at each step. We blur by convolving the current layer in the stack by a Gaussian filter.

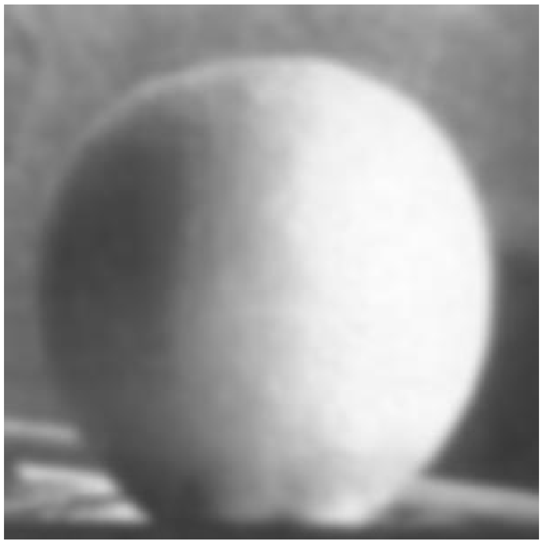
Laplacian Stack

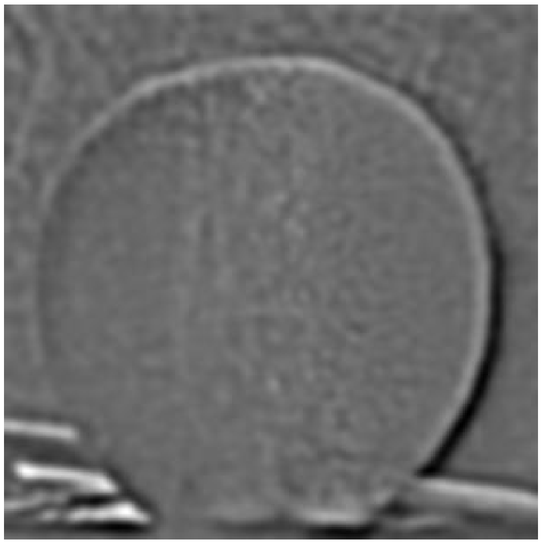
A Laplacian Stack is a collection of same-sized images made by finding the high frequencies that exist in the current layer of the Gaussian Stack, but not in any of the layers below the stack. To find layer i of the Laplacian Stack, you subtract the i - 1th layer of the Gaussian Stack from the ith layer of the Gaussian Stack. At the last layer of the Laplacian Stack, you save the most blurred version of the input image you have in the Gaussian Stack.

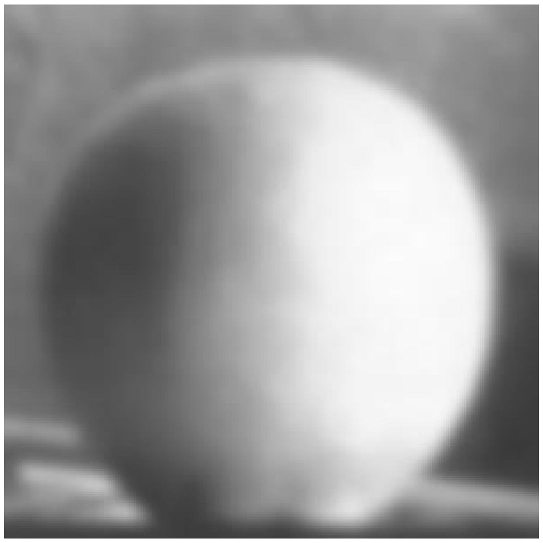
Here's what the Gaussian and Laplacian stacks for the red channel of an orange looks like:

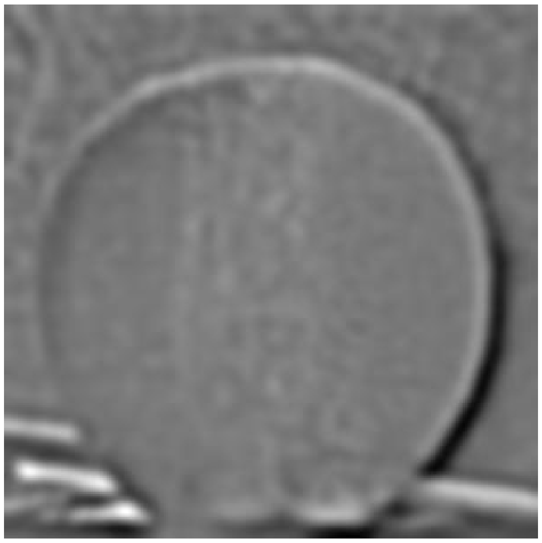

Gaussian Stack Level 0

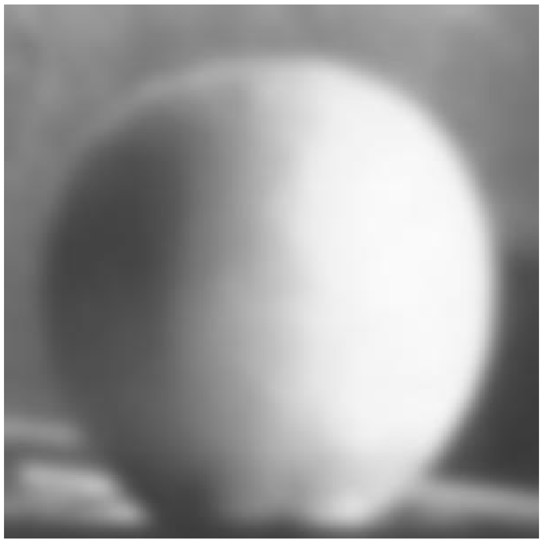

Laplacian Stack Level 0

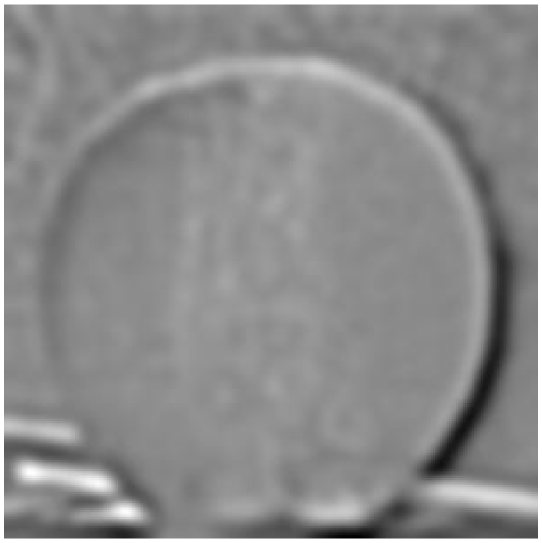

Gaussian Stack Level 1

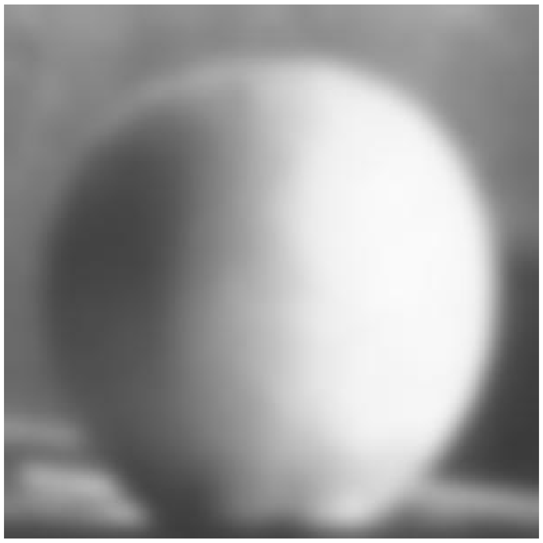

Laplacian Stack Level 1

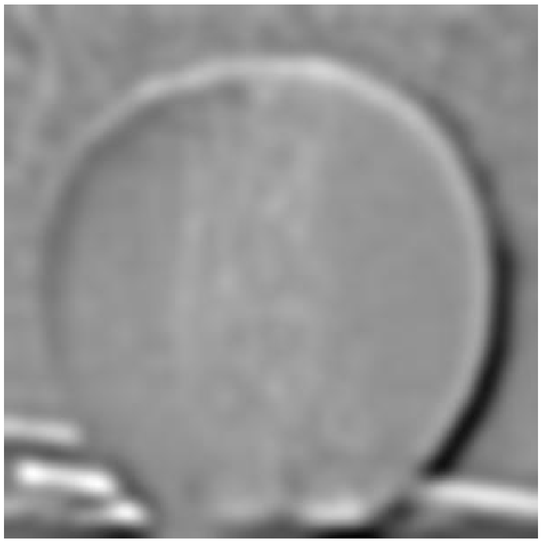

Gaussian Stack Level 2


Laplacian Stack Level 2

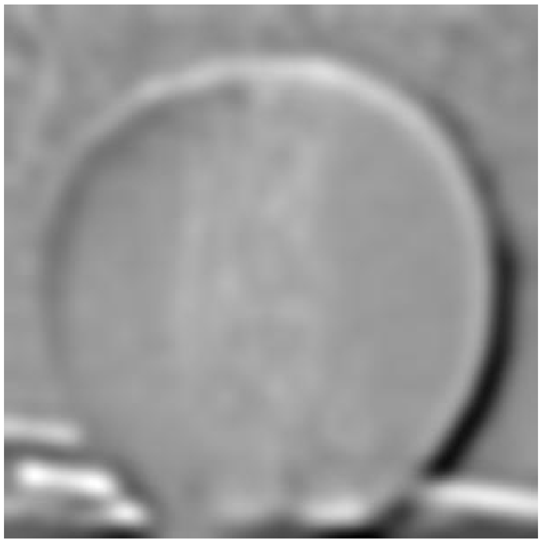

Gaussian Stack Level 3


Laplacian Stack Level 3

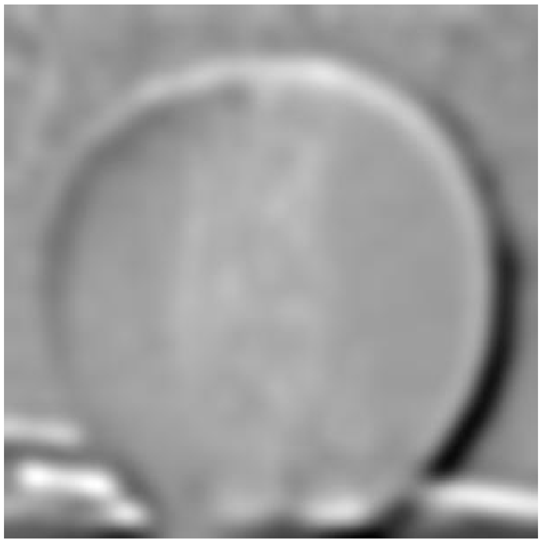

Gaussian Stack Level 4


Laplacian Stack Level 4

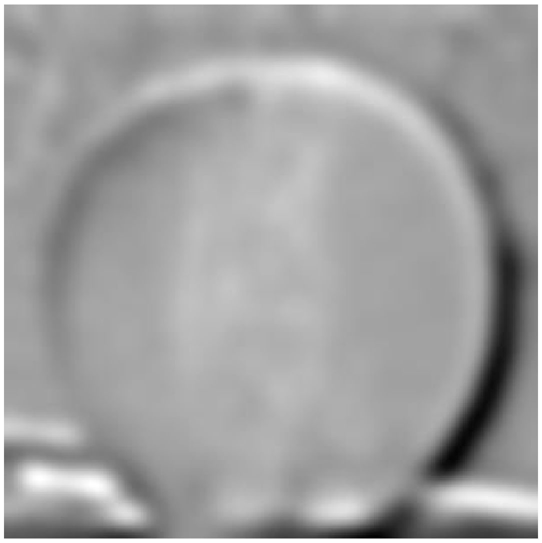
Gaussian Stack Level 5


Laplacian Stack Level 5

Gaussian Stack Level 6


Laplacian Stack Level 6

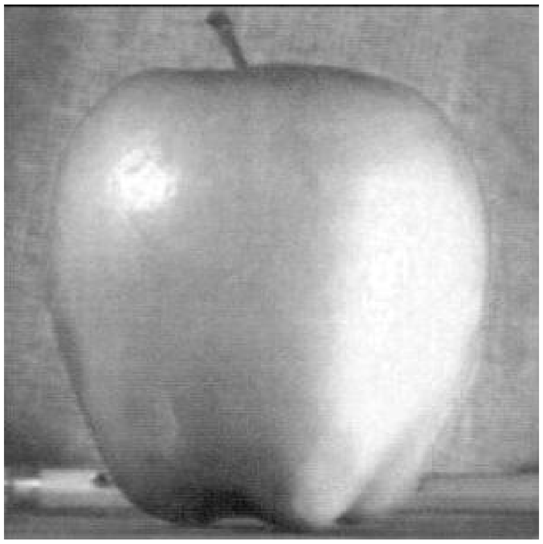
Gaussian Stack Level 7

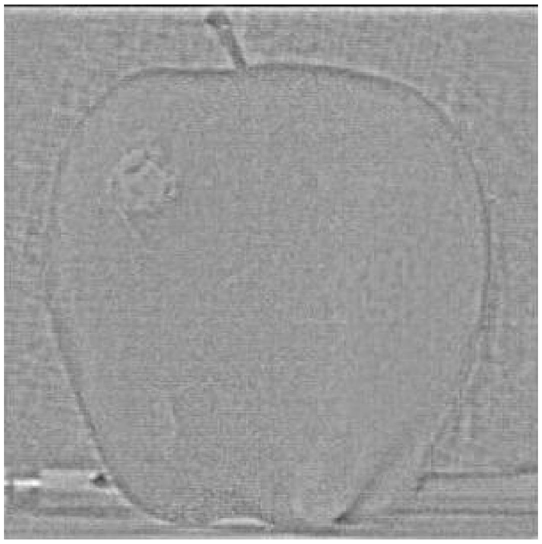

Laplacian Stack Level 7

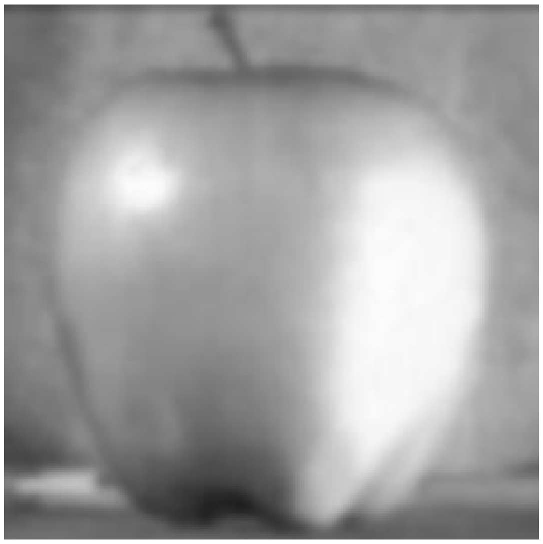
Gaussian Stack Level 8

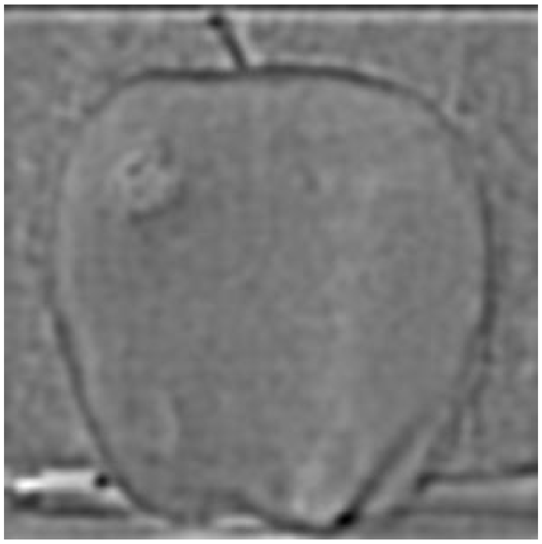
Laplacian Stack Level 8

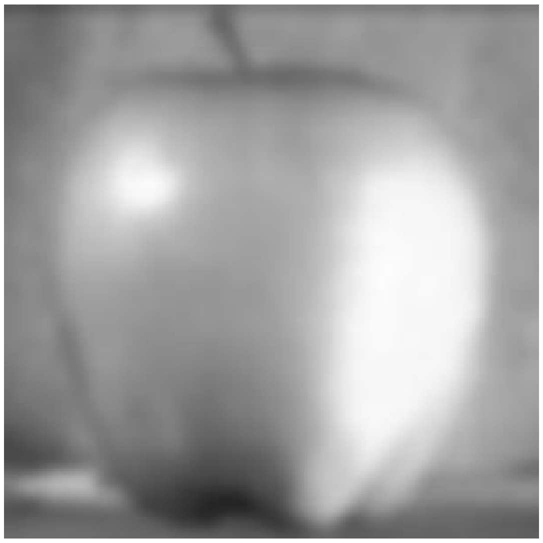
Here's what the Gaussian and Laplacian stacks for the red channel of an apple looks like

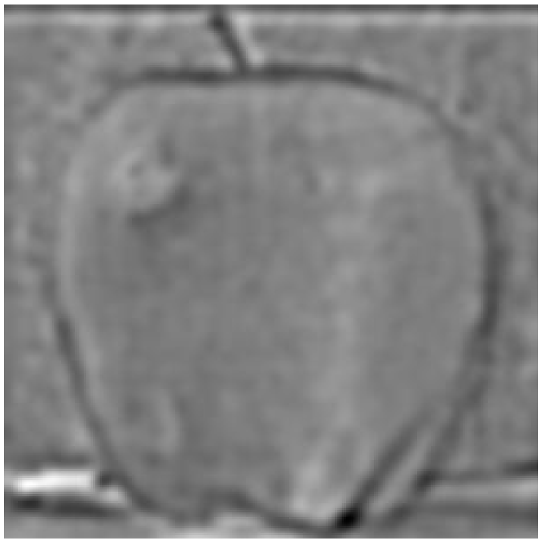

Gaussian Stack Level 0

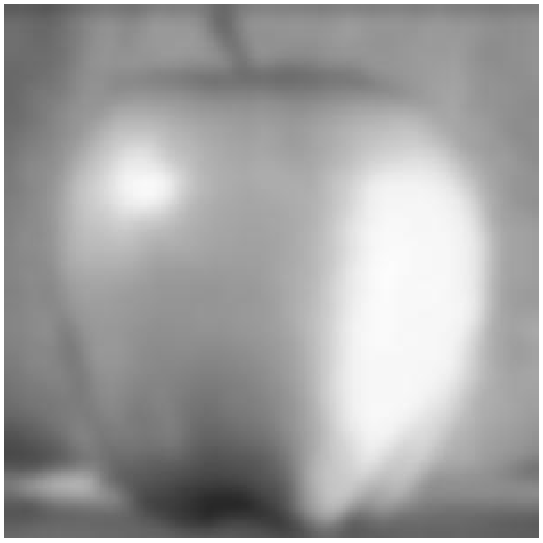

Laplacian Stack Level 0

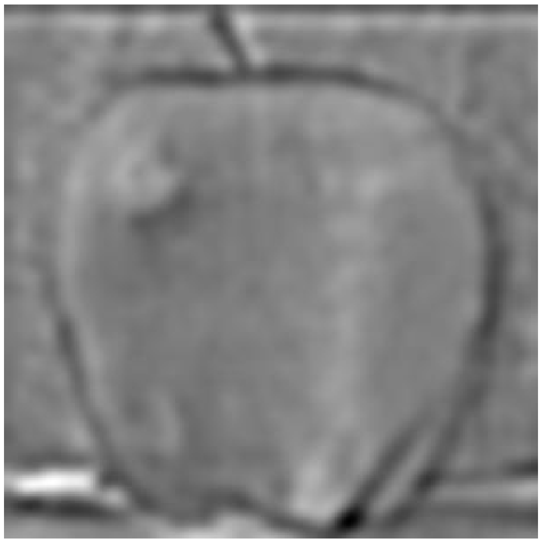

Gaussian Stack Level 1

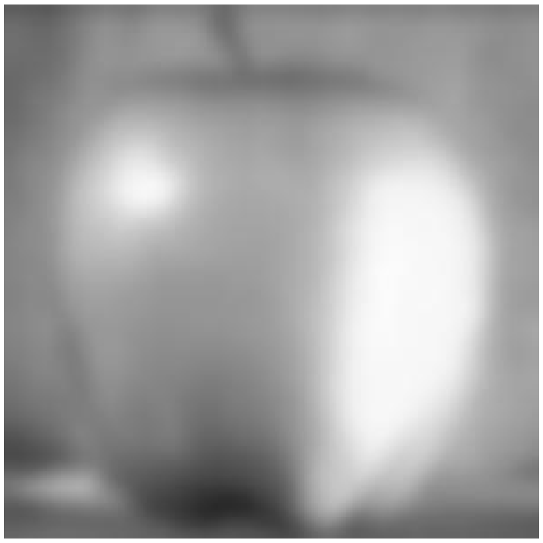

Laplacian Stack Level 1

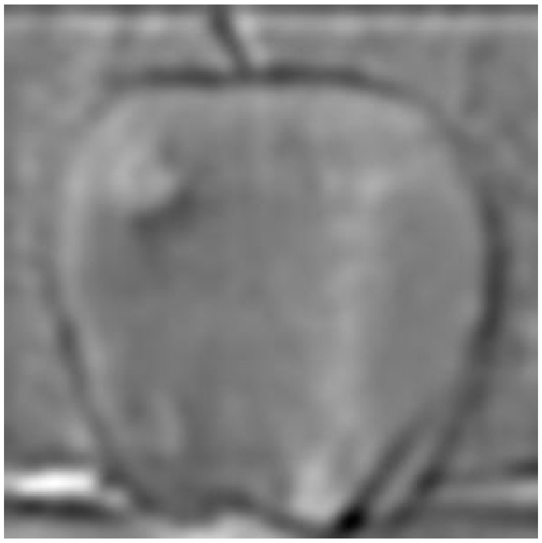

Gaussian Stack Level 2


Laplacian Stack Level 2

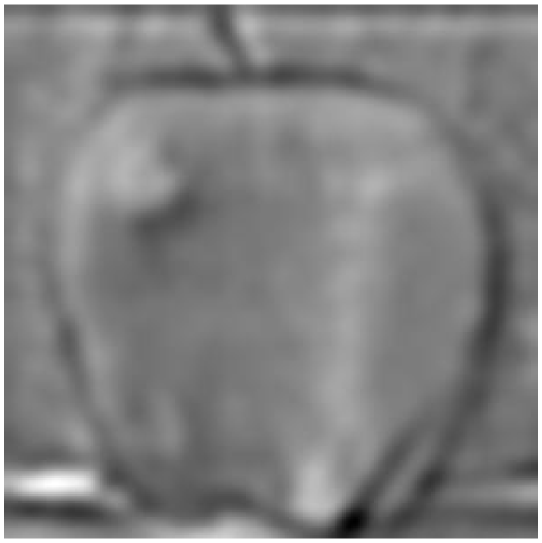

Gaussian Stack Level 3


Laplacian Stack Level 3

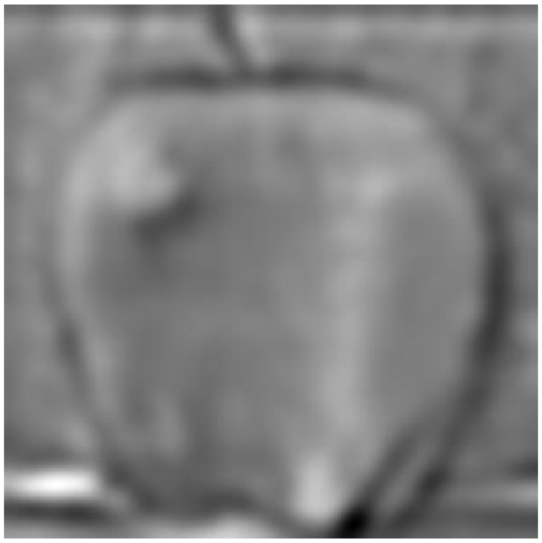

Gaussian Stack Level 4


Laplacian Stack Level 4

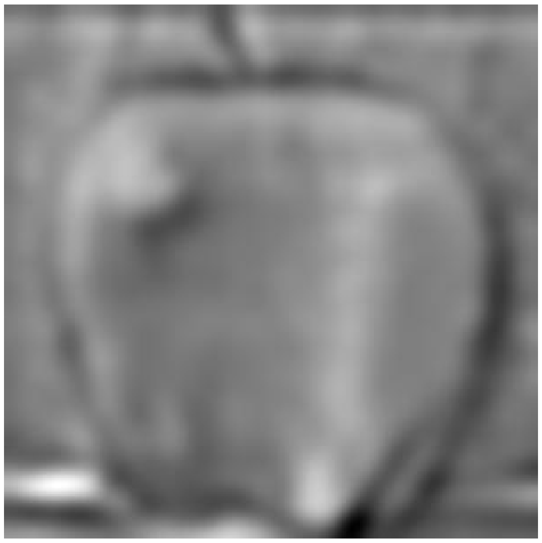
Gaussian Stack Level 5


Laplacian Stack Level 5

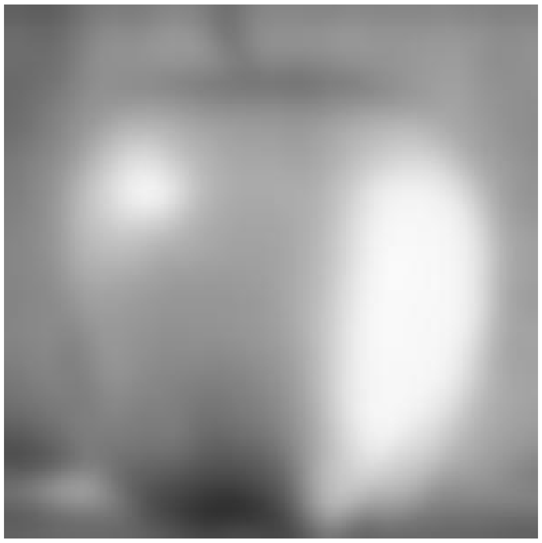
Gaussian Stack Level 6


Laplacian Stack Level 6

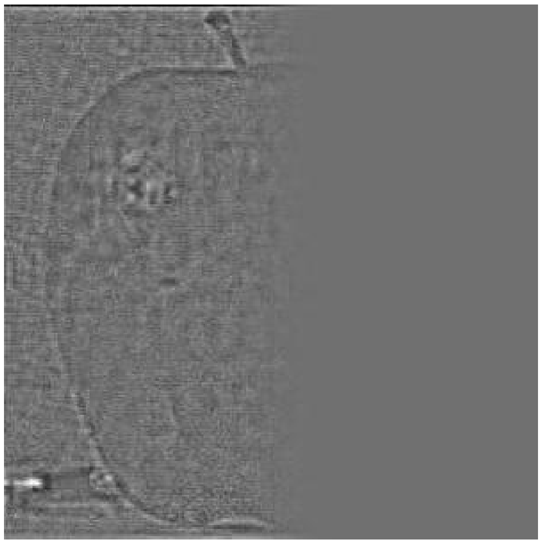
Gaussian Stack Level 7

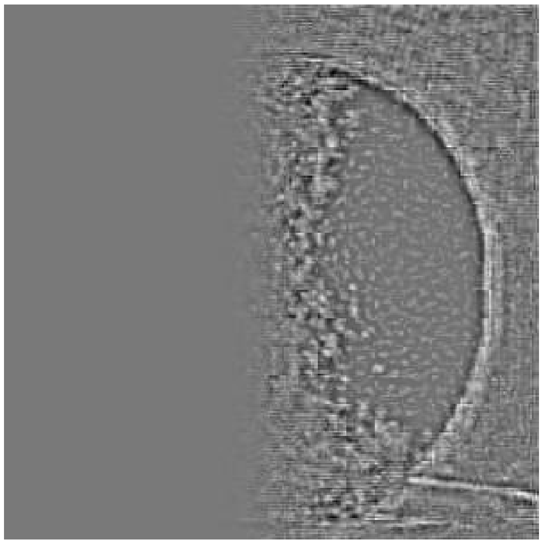

Laplacian Stack Level 7

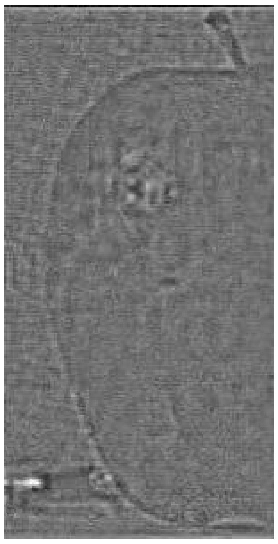
Gaussian Stack Level 8

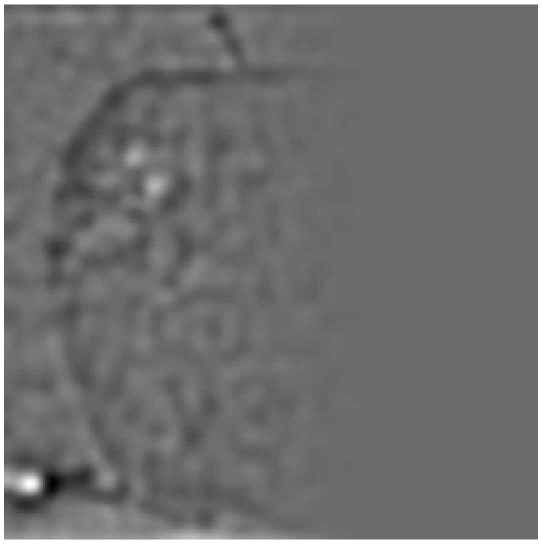

Laplacian Stack Level 8

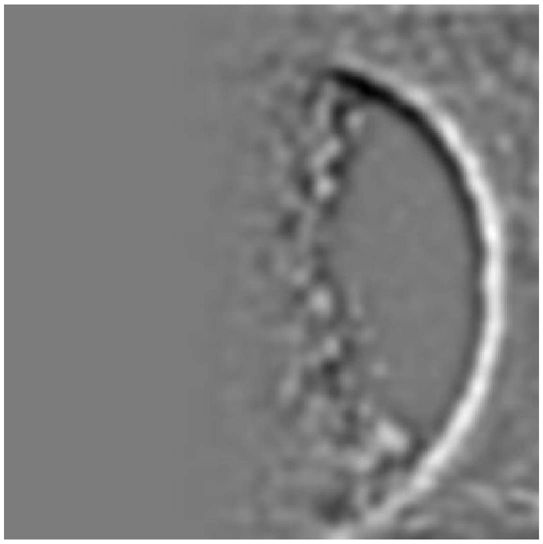
Here's we can see how we can use two images and their laplacian stacks to make a mixed image like an orapple:

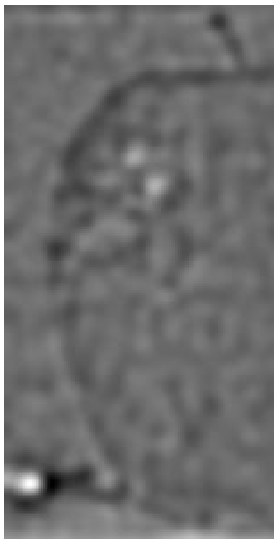

Apple Laplacian Stack Level 0 (Red Channel)

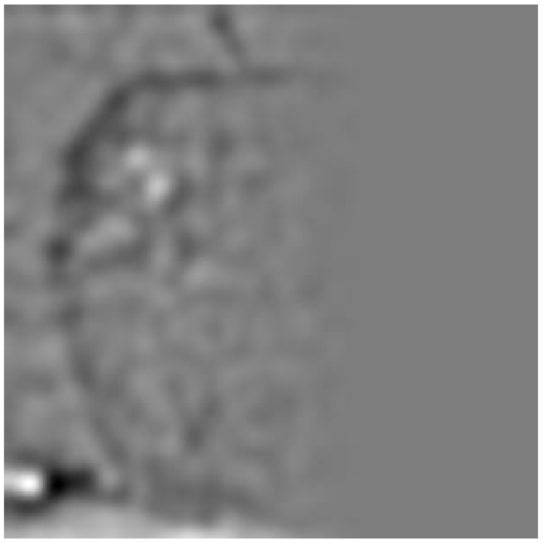

Orange Laplacian Stack Level 0 (Red Channel)

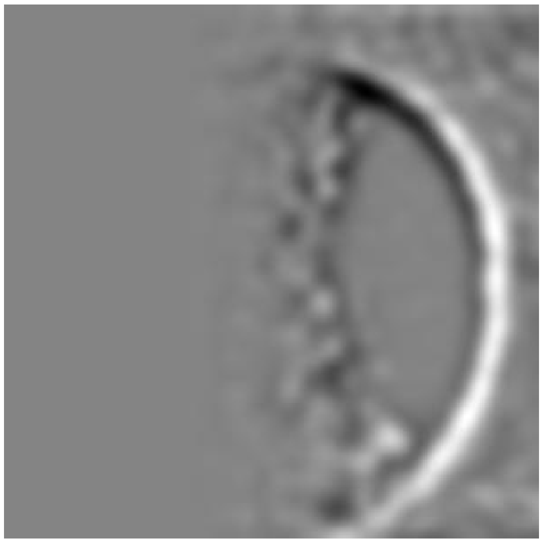

Orapple Laplacian Stack

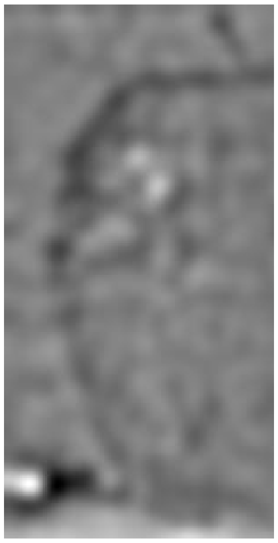

Apple Laplacian Stack Level 2 (Red Channel)

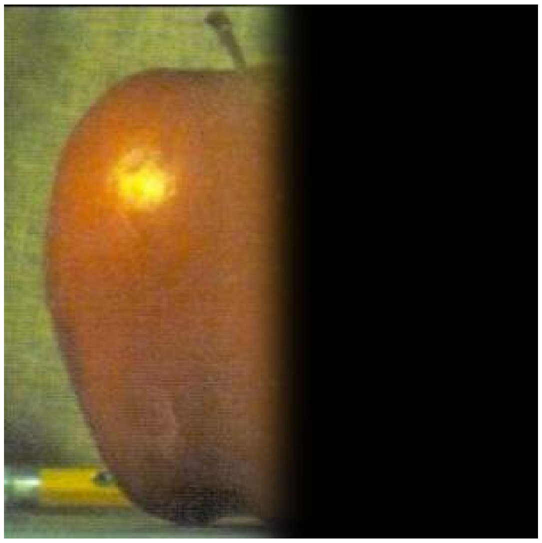

Orange Laplacian Stack Level 2 (Red Channel)

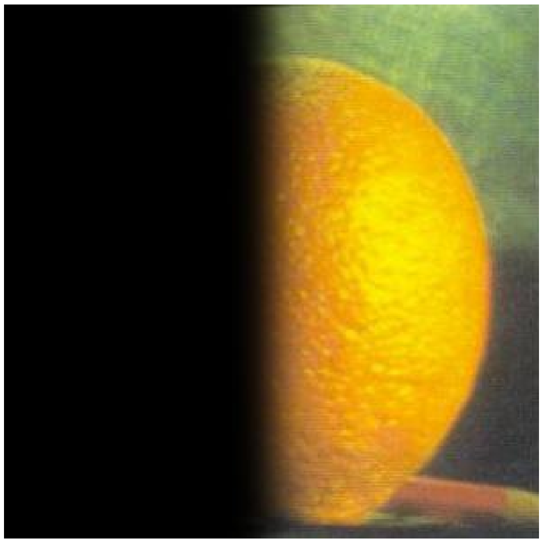

Orapple Laplacian Stack

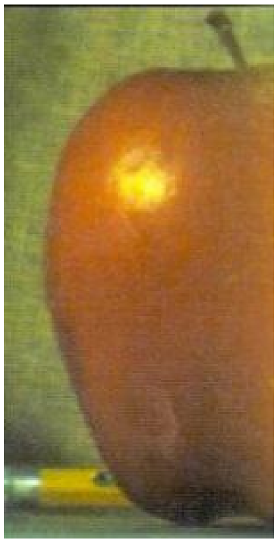

Apple Laplacian Stack Level 4 (Red Channel)


Orange Laplacian Stack Level 4 (Red Channel)


Orapple Laplacian Stack


Combined Apple Laplacian Stack (Red, Green, and Blue Channels added together)


Combined Orange Laplacian Stack (Red, Green, and Blue Channels added together)

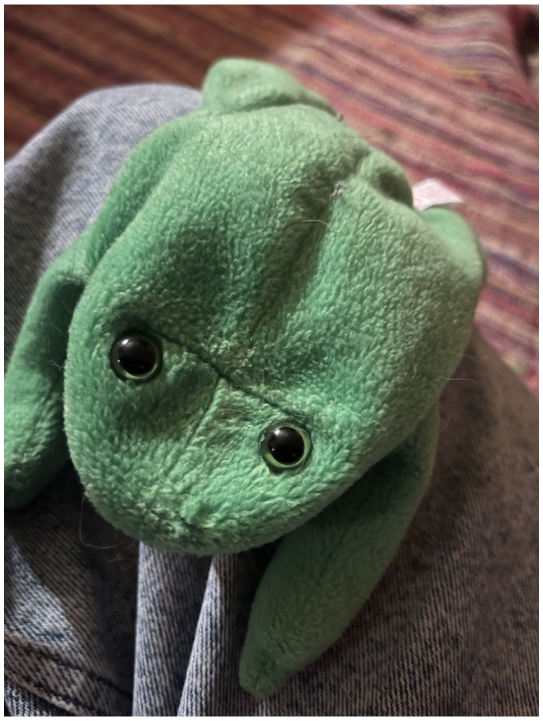

Final Orapple result (Red, added tc

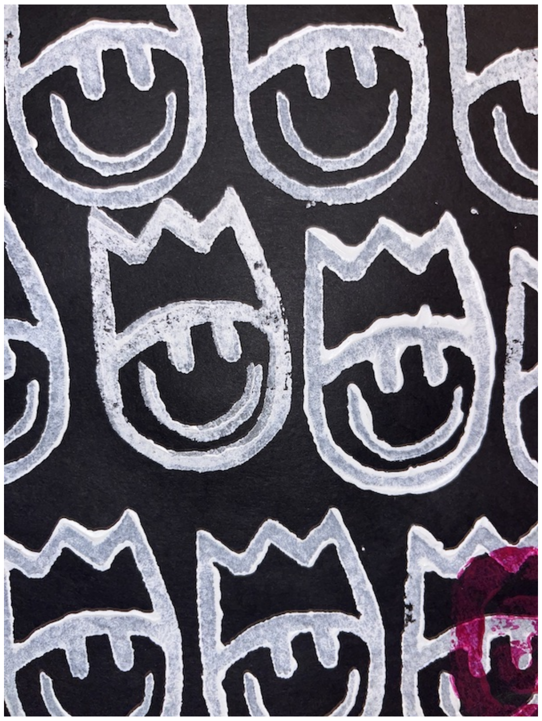
Part 2.4: Multiresolution Blending (a.k.a. the oraple!)

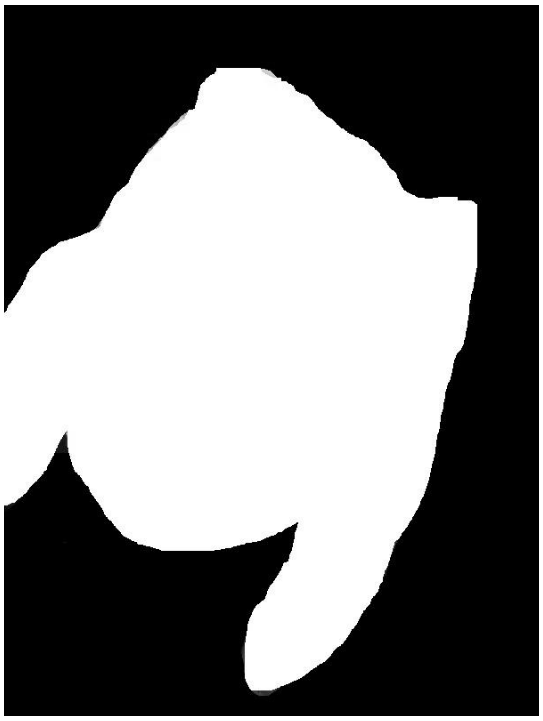
As you can see, our image blending algorithm has nice results! I achieved these results by blurring the mask I use to combine images more and more as we go from blending high frequencies to blending low frequencies. As a result, the small features contained in high frequency data is blended together over a short distance while the large features contained in low frequency data is bleneded together over long distances.

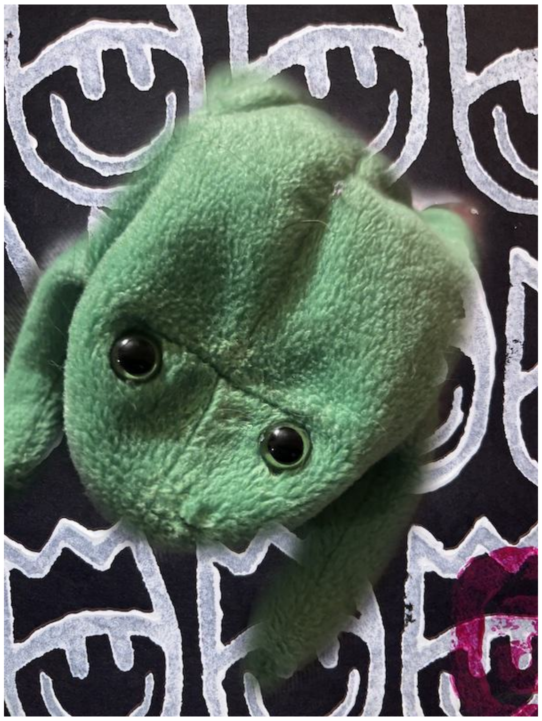
The capabilities of this algorithm is best seen when I create an irregular mask when blending two images:

Putting a frog in a world of smiley faces:


Input Image 1


Input Image 2

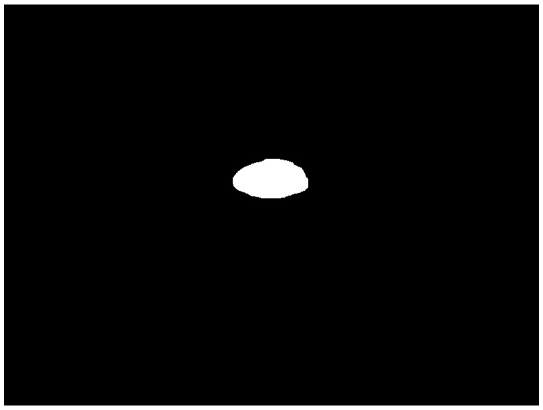

Irregular Blending Mask

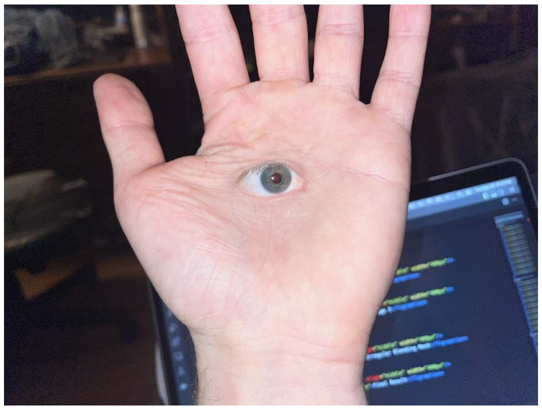

Final Result

Blending my eye into my hand:

Input Image 1

Input Image 2


Irregular Blending Mask


Final Result

Bells & Whistles

I implemented full color for blending when completing this assignment

The coolest part of this assignment is finding out that I could edit frequencies at specific frequency bands using Laplacian Stacks. I want make a video where the starting frame has the high frequencies of the first video and low frequencies of the second video, then as time goes on the frequencies interpolate so that the ending frame has low frequencies of the first video and high frequencies of the second video.

https://inst.eecs.berkeley.edu/~cs194-26/fa22/upload/files/proj1/cs194-26-ahn/