



Programming Project #2 (proj2)

CS180: Intro to Computer Vision and Computational Photography

DUE DATE: 11:59PM ON FRIDAY, SEP 26, 2025 [START EARLY]

Fun with Filters and Frequencies!

Important Note: This project requires you to show many image results. We suggest using medium-size images (less than 0.8 MB per image) as your testing cases for all questions in this project.

Part 1: Fun with Filters

In this part, we will build intuitions about 2D convolutions and filtering.

We will begin by using the humble finite difference as our filter in the x and y directions.

$$D_x = [1, 0, -1], \quad D_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

PART 1.1: CONVOLUTIONS FROM SCRATCH!

First, let's recap what a convolution is. Implement it with four for loops, then two for loops. Implement padding, with zero fill values; convolution without padding will receive partial credit. Compare it with a built-in convolution function [scipy.signal.convolve2d](#)! Then, take a picture of yourself (and read it as grayscale), write out a 9x9 box filter, and convolve the picture with the box filter. Do it with the finite difference operators D_x and D_y as well. Include the code snippets in the website!

What can you use for this section? This section is meant to be done with numpy only, simple array operations.

PART 1.2: FINITE DIFFERENCE OPERATOR

First, show the partial derivative in x and y of the [cameraman image](#) by convolving the image with finite difference operators D_x and D_y . Now compute and show the gradient magnitude image. To turn this into an edge image, let's binarize the gradient magnitude image by picking the appropriate threshold (trying to suppress the noise while showing all the real edges; it will take you a few tries to find the right threshold; This threshold is meant to be assessed qualitatively). You can use [scipy.signal.convolve2d](#).

PART 1.3: DERIVATIVE OF GAUSSIAN (DOG) FILTER

We noted that the results with just the difference operator were rather noisy. Luckily, we have a smoothing operator handy: the Gaussian filter G . Create a blurred version of the original image by convolving with a gaussian and repeat the procedure in the previous part (one way to create a 2D gaussian filter is by using `cv2.getGaussianKernel()` to create a 1D gaussian and then taking an outer product with its transpose to get a 2D gaussian kernel).

- What differences do you see?

Now we can do the same thing with a single convolution instead of two by creating a derivative of gaussian filters. Convolve the gaussian with D_x and D_y and display the resulting DoG filters as images.

- Verify that you get the same result as before.

BELLS & WHISTLES (EXTRA FOR CS180, MANDATORY FOR CS280A)

Compute the image gradient orientations, and visualize it with HSV color space!

Hint: Use hue to visualize the gradient orientation, perhaps using one of the [cyclic colormaps in matplotlib](#).

Part 2: Fun with Frequencies!

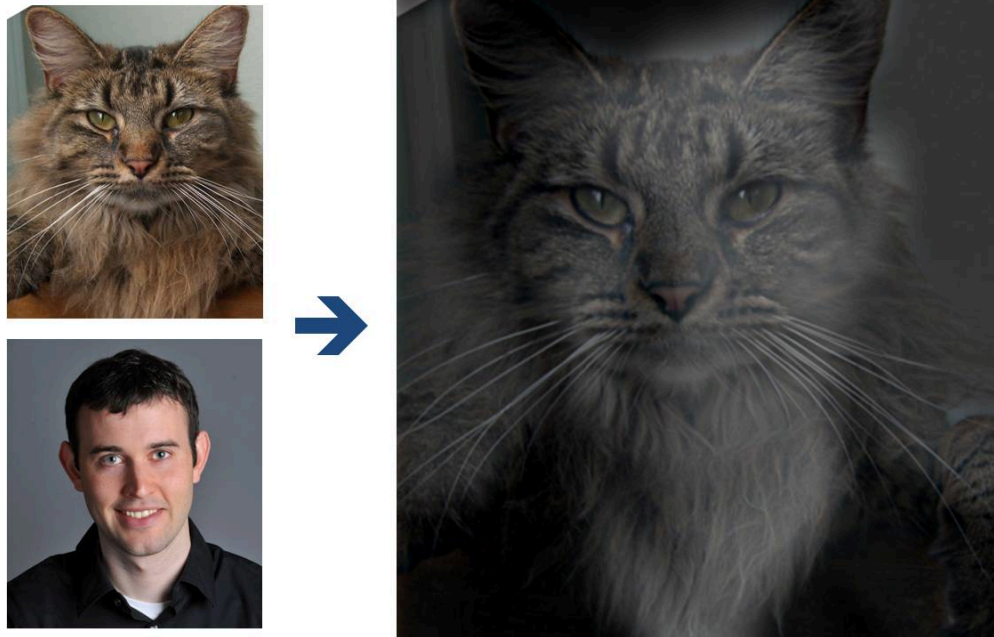
PART 2.1: IMAGE "SHARPENING"

Pick your favorite blurry image and get ready to "sharpen" it! We will derive the unsharp masking technique. Remember our favorite Gaussian filter from class. This is a low pass filter that retains only the low frequencies. We can subtract the blurred version from the original image to get the high frequencies of the image. An image often looks sharper if it has stronger high frequencies. So, let's add a little bit more high frequencies to the image! Combine this into a single convolution operation which is called the unsharp mask filter. Show your result on the following image (download [here](#)) plus other images of your choice --



Also for evaluation, pick a sharp image, blur it and then try to sharpen it again. Compare the original and the sharpened image and report your observations.

PART 2.2: HYBRID IMAGES



(Look at image on right from very close, then from far away.)

OVERVIEW

The goal of this part of the assignment is to create [hybrid images](#) using the approach described in the SIGGRAPH 2006 [paper](#) by Oliva, Torralba, and Schyns. *Hybrid images* are static images that change in interpretation as a function of the viewing distance. The basic idea is that high frequency tends to dominate perception when it is available, but, at a distance, only the low frequency (smooth) part of the signal can be seen. By blending the high frequency portion of one image with the low-frequency portion of another, you get a hybrid image that leads to different interpretations at different distances.

DETAILS

[Here](#), we have included two sample images (of Derek and his former cat Nutmeg) and some starter code that can be used to load two images and align them. The alignment is important because it affects the perceptual grouping (read the paper for details).

1. First, you'll need to get a few pairs of images that you want to make into hybrid images. You can use the sample images for debugging, but you should use your own images in your results. Then, you will need to write code to low-pass filter one image, high-pass filter the second image, and add (or average) the two images. For a low-pass filter, Oliva et al. suggest using a standard 2D Gaussian filter. For a high-pass filter, they suggest using the impulse filter minus the Gaussian filter (which can be computed by subtracting the Gaussian-filtered image from the original). The [cutoff-frequency](#) of each filter should be chosen with some experimentation.
2. For your favorite result, you should also illustrate the process through frequency analysis. Show the log magnitude of the Fourier transform of the two input images, the filtered images, and the hybrid image. In Python, you can compute and display the 2D Fourier transform with:

```
plt.imshow(np.log(np.abs(np.fft.fftshift(np.fft.fft2(gray_image)))))
```
3. Try creating 2-3 hybrid images (change of expression, morph between different objects, change over time, etc.). Show the input image and hybrid result per example. (No need to show the intermediate results as in step 2.)

BELLS & WHISTLES (EXTRA FOR CS180, MANDATORY FOR CS280A)

Try using color to enhance the effect. Does it work better to use color for the high-frequency component, the low-frequency component, or both?

MULTI-RESOLUTION BLENDING AND THE ORAPLE JOURNEY

OVERVIEW

The goal of this part of the assignment is to blend two images seamlessly using a multi resolution blending as described in the 1983 [paper](#) by Burt and Adelson. An *image spline* is a smooth seam joining two image together by gently distorting them. *Multiresolution blending* computes a gentle seam between the two images seperately at each band of image frequencies, resulting in a much smoother seam.

We'll approach this section in two steps:

1. creating and visualizing the Gaussian and Laplacian stacks and
2. blending together images with the help of the completed stacks, and exploring creative outcomes

PART 2.3: GAUSSIAN AND LAPLACIAN STACKS

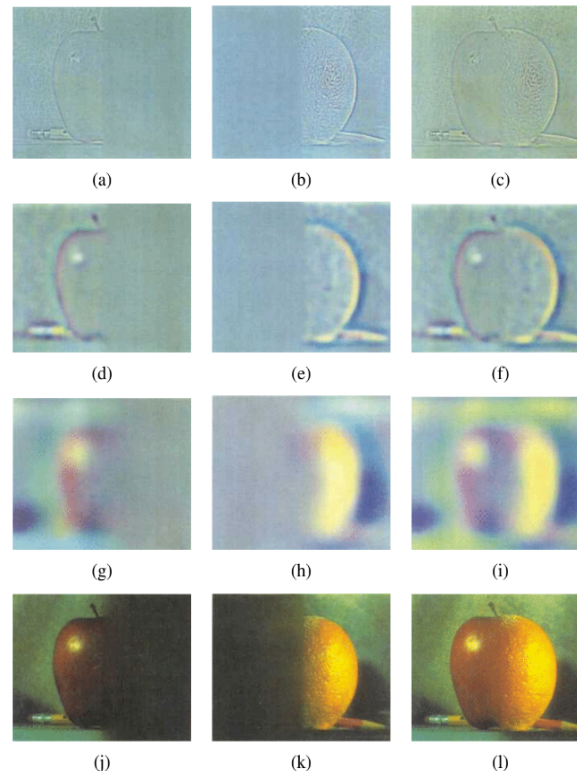


Figure 3.42 Laplacian pyramid blending details (Burt and Adelson 1983b) © 1983 ACM. The first three rows show the high, medium, and low-frequency parts of the Laplacian pyramid (taken from levels 0, 2, and 4). The left and middle columns show the original apple and orange images weighted by the smooth interpolation functions, while the right column shows the averaged contributions.

OVERVIEW

In this part you will implement Gaussian and Laplacian stacks, which are kind of like pyramids but without the downsampling. This will prepare you for the next step for Multi-resolution blending.

DETAILS

1. Implement a Gaussian and a Laplacian **stack**. The different between a stack and a pyramid is that in each level of the pyramid the image is downsampled, so that the result gets smaller and smaller. In a stack the images are never downsampled so the results are all the same dimension as the original image, and can all be saved in one 3D matrix (if the original image was a grayscale image). To create the

successive levels of the Gaussian Stack, just apply the Gaussian filter at each level, but do not subsample. In this way we will get a stack that behaves similarly to a pyramid that was downsampled to half its size at each level. If you would rather work with pyramids, you may implement pyramids other than stacks. However, in any case, you are **NOT** allowed to use built-in pyramid functions like `cv2.pyrDown()` or `skimage.transform.pyramid_gaussian()` in this project. You must implement your stacks from scratch!

2. Apply your Gaussian and Laplacian stacks to the Orapple and recreate the outcomes of Figure 3.42 in [Szelski \(Ed 2\)](#), page 167, as you can see in the image above. Review the 1983 [paper](#) for more information.

PART 2.4: MULTIREOLUTION BLENDING (A.K.A. THE ORAPPLE!)



OVERVIEW

Review the 1983 [paper](#) by Burt and Adelson, if you haven't! This will provide you with the context to continue. In this part, we'll focus on actually blending two images together.

DETAILS

[Here](#), we have included the two sample images from the paper (of an apple and an orange).

1. First, you'll need to get a few pairs of images that you want blend together with a vertical or horizontal seam. You can use the sample images for debugging, but you should use your own images in your results. Then you will need to write some code in order to use your Gaussian and Laplacian stacks from part 2 in order to blend the images together. Since we are using stacks instead of pyramids like in the paper, the algorithm described on page 226 will not work as-is. If you try it out, you will find that you end up with a very clear seam between the apple and the orange since in the pyramid case the downsampling/blurring/upsampling hoopla ends up blurring the abrupt seam proposed in this algorithm. Instead, you should always use a mask as is proposed in the algorithm on page 230, and remember to create a Gaussian stack for your mask image as well as for the two input images. The Gaussian blurring of the mask in the pyramid will smooth out the transition between the two images. For the vertical or horizontal seam, your mask will simply be a step function of the same size as the original images.
2. Now that you've made yourself an orapple (a.k.a your vertical or horizontal seam is nicely working), pick two pairs of images to blend together with an irregular mask, as is demonstrated in figure 8 in the paper.
3. Blend together some crazy ideas of your own!
4. Illustrate the process by applying your Laplacian stack and displaying it for your favorite result and the masked input images that created it. This should look similar to Figure 10 in the paper.

BELLS & WHISTLES (EXTRA FOR CS180, MANDATORY FOR CS280A)

- Try using color to enhance the effect.

DELIVERABLES

For this project you must turn in both your code and a project webpage as described [here](#) and tell us about the most important thing you learned from this project!

Also, submit the webpage URL to the class gallery via Google Form [here](#).

The detailed list of deliverables is released here for transparency. This is what we will check for in your website submission! However, we reserve the right to grade on a case to case basis, and we will *not* release the point distribution for each item.

Part 1: Filters and Edges

- **Part 1.1.** Provide correct implementations of convolution, with numpy only. Compare with `scipy.signal.convolve2d`, and comment on runtime and how boundaries are handled.
- **Part 1.2.** Show partial derivatives in x and y, the gradient magnitude image, and a binarized edge image. Feel free to make minor tradeoffs between finding all edges and removing all noise, but please justify your decision.
- **Part 1.3.** Construct Gaussian filters using `cv2.getGaussianKernel`, build DoG filters, and visualize them. Apply both Gaussian smoothing and DoG filters to the cameraman image. Compare these results with those from the finite difference method.
- **Bells and whistles.** Compute and visualize gradient orientations for the cameraman image (without using built-in angle functions) and explain your process.

Part 2: Applications

- **Part 2.1.** Implement the unsharp mask filter. Explain how it works in relation to blur filters and high frequencies. Show the blurred, high-frequency, and sharpened versions of the Taj Mahal image, as well as another image of your choice. Demonstrate how varying the sharpening amount changes the result.
- **Part 2.2.** Create three hybrid images (including Derek + Nutmeg and two of your own). For one hybrid, show the entire process: original and aligned images, Fourier transforms, filtered results, cutoff frequency choice, and final image. For the others, present the originals and the final hybrid.
- **Part 2.2 (bells and whistles)** Explore, and justify your choices!
- **Part 2.3 + 2.4** Visualize the process (show the Gaussian and Laplacian stacks for the Orange + Apple images, and recreate the outcomes of Figure 3.42 (a) through (l)). Also, include 2 additional custom blended images; one of them should have an irregular mask (not just straight lines).
- **Part 2.4 (bells and whistles)** Explore, and justify your choices!

SCORING

CS180

The first part of the assignment is worth **30 points**. The following things need to be answered in the html webpage *along* with the **visualizations** mentioned in the problem statement. The distribution is as follows:

- **(10 points)** Part 1.1: Convolutions from Scratch!
- **(10 points)** Part 1.2: Finite Difference Operator
- **(10 points)** Part 1.3: Derivative of Gaussian (DoG) Filter

The second part of the assignment is worth **65 points**, as follows:

- **(15 points)** Part 2.1: Image "Sharpening"
- **(20 points)** Part 2.2: Hybrid Images
- **(10 points)** Part 2.3: Gaussian and Laplacian Stacks
- **(20 points)** Part 2.4: Multiresolution Blending

5 points for clarity. This includes the quality of the visualizations, the clarity of the explanations, and the overall organization of the webpage.

CS280A

The first part of the assignment is worth **30 points**. The following things need to be answered in the html webpage *along* with the **visualizations** mentioned in the problem statement. The distribution is as follows:

- **(5 points)** Part 1.1: Convolutions from Scratch!
- **(5 points)** Part 1.2: Finite Difference Operator
- **(10 points)** Part 1.3: Derivative of Gaussian (DoG) Filter
- **(10 points) Bells and whistles - Mandatory!**

The second part of the assignment is worth **65 points**, as follows:

- **(15 points)** Part 2.1: Image "Sharpening"
- **(20 points)** Part 2.2: Hybrid Images -- **Bells and whistles section is mandatory!**
- **(10 points)** Part 2.3: Gaussian and Laplacian Stacks
- **(20 points)** Part 2.4: Multiresolution Blending -- **Bells and whistles section is mandatory!**

5 points for clarity. This includes the quality of the visualizations, the clarity of the explanations, and the overall organization of the webpage.

ACKNOWLEDGEMENTS

The hybrid images part of this assignment is borrowed from Derek Hoiem's [Computational Photography class](#).