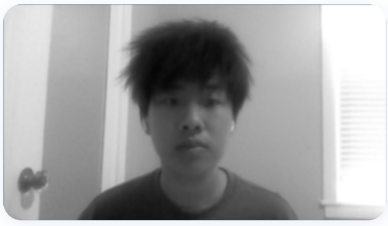
Fun with Filters and Frequencies!

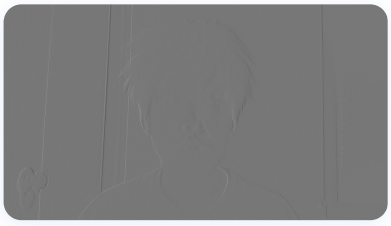
Part 1: Fun with Filters

Part 1.1: Convolutions from Scratch!

Write a convolution function of 4 loops and 2 loops from scratch. Compare them with the scipy.

4 loops

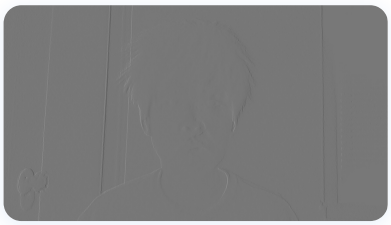

box filter


dx


dy

2 loops

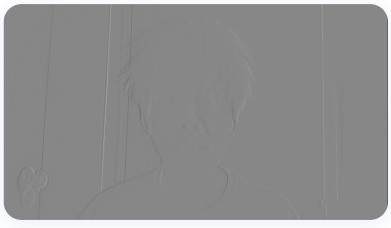

box filter

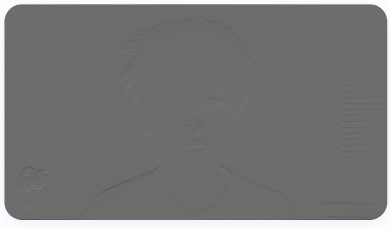

dx


dy

scipy


box filter

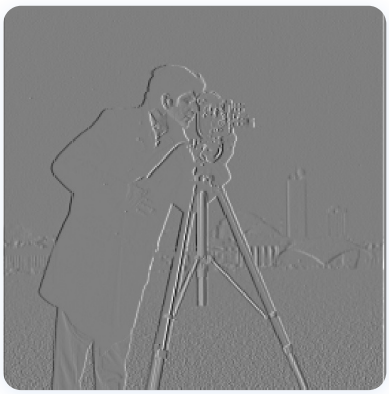

dx


dy

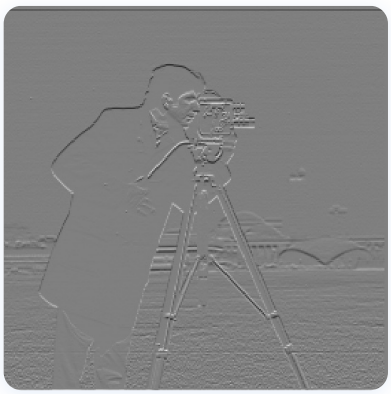
▶ Show Code

Part 1.2: Finite Difference Operator

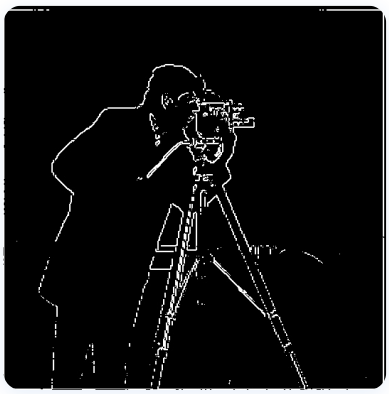
Show partial derivatives in x and y, the gradient magnitude image, and a binarized edge image.



Partial derivative in x



Partial derivative in y

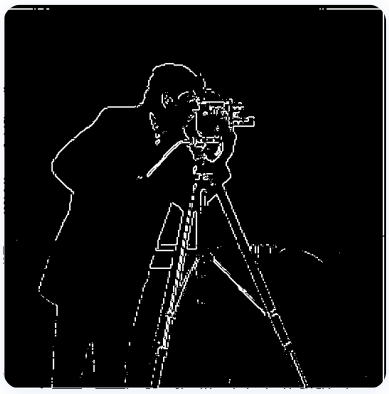


Gradient magnitude and binarized edge

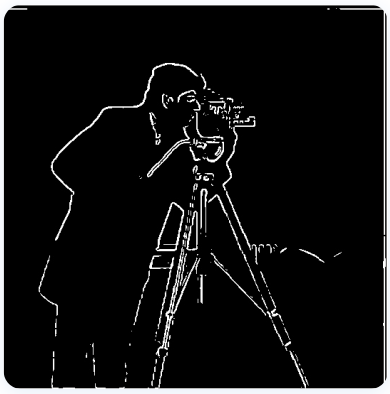
▶ Show Code

Part 1.3: Derivative of Gaussian (DoG) Filter

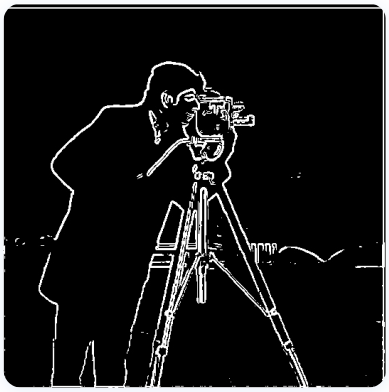
Compare the results with just the difference operator (noisy) with the DoG filter. Create DoG filters by convolving the gaussian with D_x and D_y.



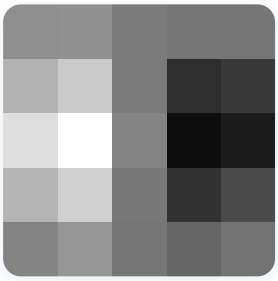
Edge with finite difference (noisy)



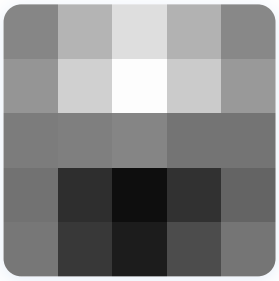
Blurred image with Gaussian



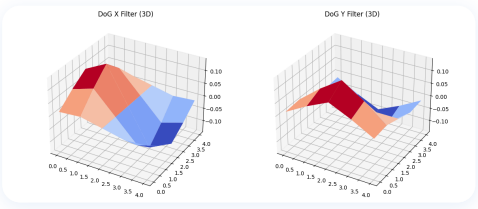
Edge with DoG filter



DoG kernel in x direction



DoG kernel in y direction



3D visualization of DoG kernels

Answer the questions:

What is the difference between the results obtained using the finite difference operator and the DoG filter?

As can be seen from the images, the edges detected using the finite difference operator (left image) contain a lot of noise, the edges are not clear, and there are many small noise points. The edges detected using the DoG filter (right image) are smoother and clearer, most of the noise has been removed, and the edge lines are more continuous and distinct. This is because the DoG filter first smooths the image with a Gaussian filter before calculating the gradient, which effectively suppresses the influence of noise.

Why does the DoG filter produce better results?

The DoG (Difference of Gaussian) filter can achieve better edge detection results for the following main reasons:

1. **Noise suppression**: The DoG filter first smooths the image using a Gaussian kernel, effectively removing noise from the image, while the finite difference operator directly calculates the gradient on the original image, which is easily disturbed by noise.
2. **Edge enhancement**: Through Gaussian filtering, the edges in the image become smoother and more continuous, and clearer edge responses can be obtained when calculating the gradient again.
3. **Biological vision inspiration**: The DoG filter simulates the mechanism of edge detection in the human visual system, which is more in line with the way the human eye perceives edges.

What is the difference between blurring and then computing the gradient and computing the gradient using the DoG kernel?

Although both methods theoretically produce the same result, there are practical differences:

1. **Computational efficiency**: Computing the gradient using the DoG kernel is more efficient as it requires only one convolution operation, while blurring and then computing the gradient requires two separate operations.
2. **Numerical precision**: Using the DoG kernel can reduce accumulated numerical errors that might occur in the two-step process.
3. **Implementation clarity**: Blurring first and then computing the gradient is more intuitive and easier to understand, while using the DoG kernel is more abstract but mathematically elegant.

▶ Show Code

Gradient Orientation Visualization

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

To visualize gradient orientations, we use the HSV color space where:

- Hue represents the gradient orientation (0° to 360° mapped to 0-180 in OpenCV)
- Saturation is set to maximum to have vivid colors
- Value is set based on gradient magnitude to show strong gradients more clearly

Gradient Orientation Visualization (HSV)

In this visualization, different colors represent different gradient orientations. Red typically represents horizontal gradients (0°), green represents 120°, and blue represents 240°. The intensity of the color corresponds to the magnitude of the gradient at that point. This provides an intuitive way to understand the distribution of edge directions in the image.

▶ Show Code

Part 2: Fun with Frequencies!

Part 2.1: Image "Sharpening"

Use a single kernel to sharpen the image. Just need to let 2 impulse filters minus the Gaussian filter.

Taj Mahal original

Taj Mahal sharpened


Chihaya Anon original

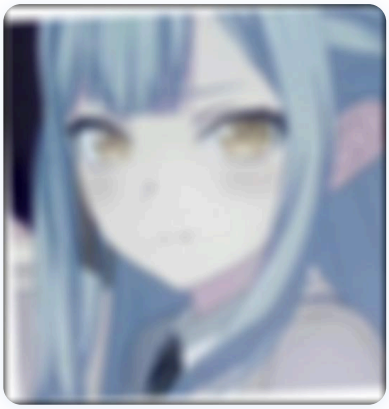

Chihaya Anon sharpened

▶ Show Code

Part 2.2: Hybrid Images

Create hybrid images using the approach described in the SIGGRAPH 2006 paper by Oliva, Torralba, and Schyns.

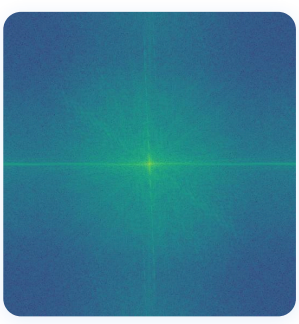
Hybrid images are static images that change in interpretation as a function of the viewing distance. The process involves combining the low-frequency content of one image with the high-frequency content of another to create a single image that can be perceived as either image depending on the viewing distance.

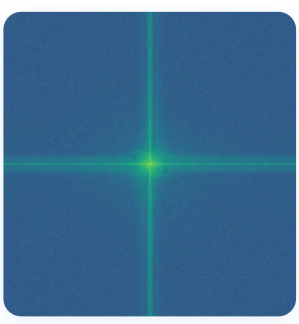

Sakiko original

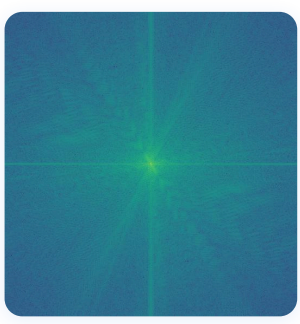

Mutsumi original

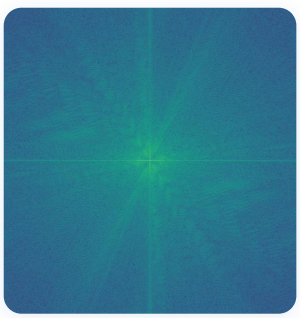

Hybrid image

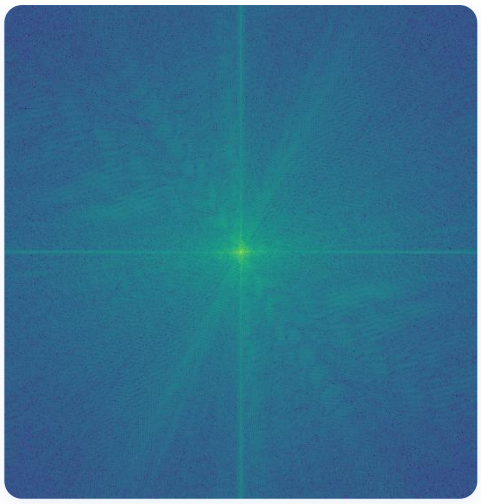
Frequency Analysis


Sakiko FFT


Sakiko low-pass FFT


Mutsumi FFT


Mutsumi high-pass FFT



Hybrid image FFT

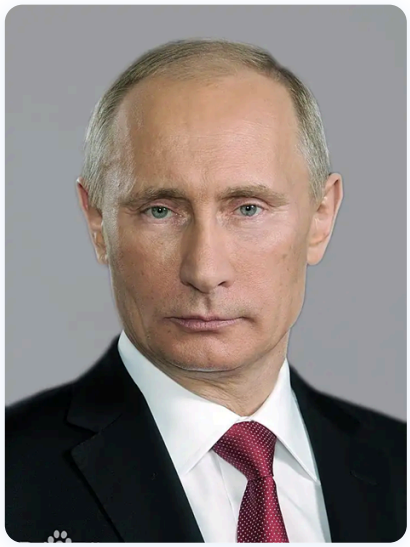
Additional Hybrid Images

Sakiko original

Misumi original

Sakiko-Misumi hybrid

Trump original



Putin original

Trump-Putin hybrid

Whitehouse original

Tiananmen original

Whitehouse-Tiananmen hybrid

How it works: The technique involves creating a low-pass filtered version of one image (preserving low-frequency content) and a high-pass filtered version of another image (preserving high-frequency content). These are then combined to form the hybrid image. When viewed up close, the high-frequency content dominates perception, while at a distance, the low-frequency content becomes more apparent.

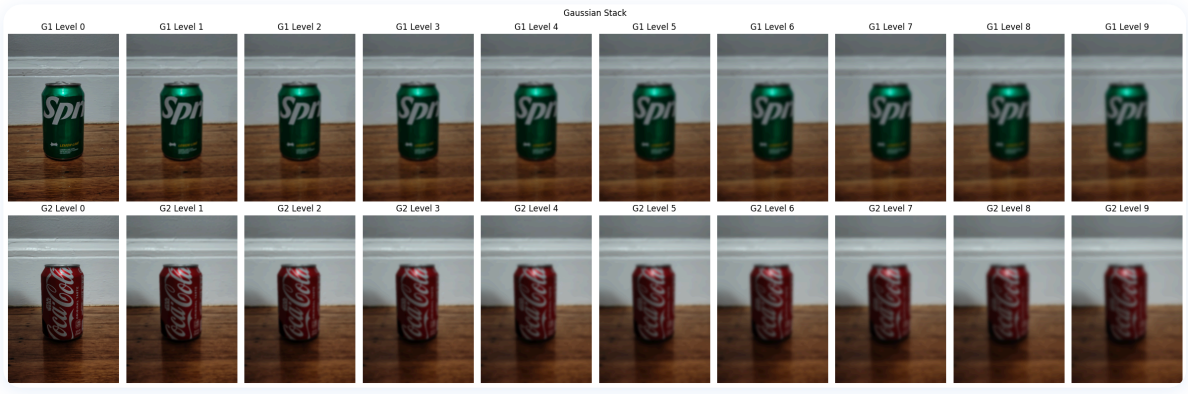
▶ Show Code

Part 3: Multi-resolution Blending

Part 3.1: Gaussian and Laplacian Stacks

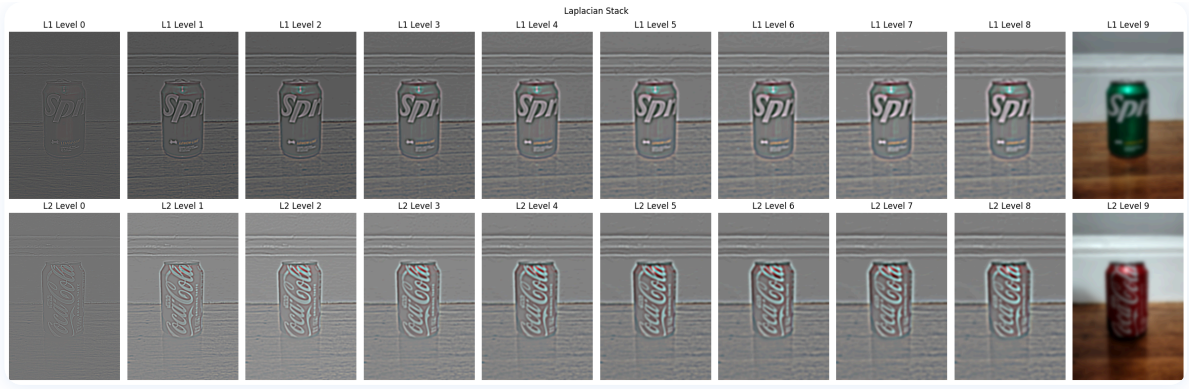
In this part we implement Gaussian and Laplacian stacks. The difference between a stack and a pyramid is that in each level of the pyramid the image is downsampled, so that the result gets smaller and smaller. In a stack the images are never downsampled so the results are all the same dimension as the original image.

Gaussian Stack



Gaussian Stack Visualization

Laplacian Stack



Laplacian Stack Visualization

▶ Show Code

Part 3.2: Blending Process

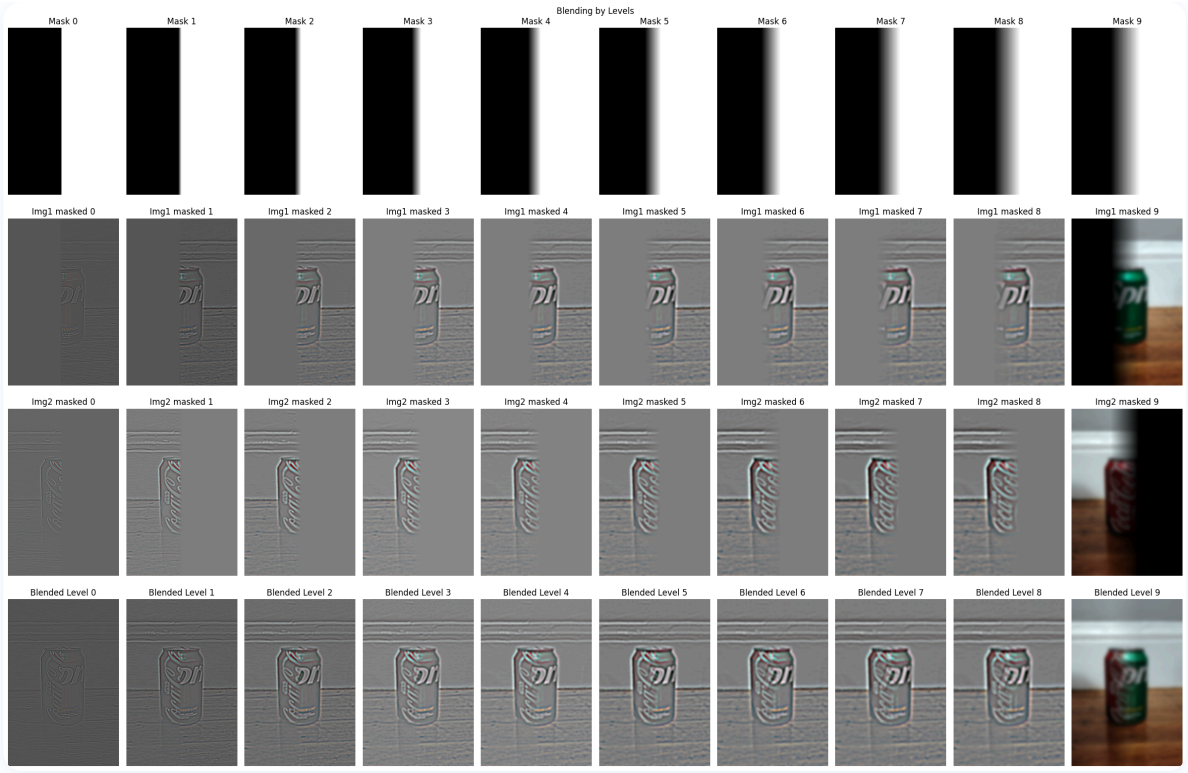
Here we demonstrate the multiresolution blending process using the Gaussian and Laplacian stacks. We blend two images (Coke and Sprite) using a mask to create a seamless transition.

Input Images

Sprite Image

Coke Image

Blending Process by Levels



Blending Process Visualization

Final Blended Image

Final Blended Image

▶ Show Code