

# Computer Vision HW2

---

Group 11

Member: 110705017 何翊華, 313551097 鄭淮薰, 313551098 張穰齡

## Introduction

---

In this assignment, we explore several important techniques in the field of image processing, specifically focusing on hybrid images, image pyramids, and colorization of historical images.

The hybrid image technique combines a low-pass filtered version of one image with a high-pass filtered version of another. This method allows the viewer to perceive different images depending on the viewing distance, exploiting the characteristics of human visual perception. By adjusting the cutoff frequency, the amount of high and low frequencies can be finely controlled, affecting the visual outcome.

In addition, we delve into the concept of image pyramids, which represent images at multiple resolutions, helping in tasks such as image blending and efficient storage. Lastly, the colorization task aims to recreate color images from Prokudin-Gorskii's glass plate negatives, requiring precise alignment of RGB channels to reconstruct a visually appealing color image. These tasks not only test the fundamental principles of image processing but also demonstrate practical applications in modern computer vision.

## Implementation Procedure

---

### 1. Hybrid image

A hybrid image  $H$  is obtained by combining two images  $I_1, I_2$ , with first image filtered with a low-pass filter  $G_1$  and the second image filtered with high-pass filter  $1 - G_2$ :  $H = I_1 \cdot G_1 + I_2 \cdot (1 - G_2)$ . The operations are defined in the Fourier domain. There are two free parameters: the cutoff-frequency of high-pass filter and low-pass filter, which controls how much high frequency to remove from the first image and how much low frequency to leave in the second image.

Here, we apply two different filters to generate hybrid images, gaussian filter and ideal filter as follow:

1. Ideal filter

- High-pass  $H(u, v) = \begin{cases} 0 & \text{if } D(u, v) \leq D_0 \\ 1 & \text{if } D(u, v) > D_0 \end{cases}$   
where  $D_0$  is cutoff frequency,  $D(u, v) = \sqrt{u^2 + v^2}$
- Low-pass  $H(u, v) = \begin{cases} 1 & \text{if } D(u, v) \leq D_0 \\ 0 & \text{if } D(u, v) > D_0 \end{cases}$   
where  $D_0$  is cutoff frequency,  $D(u, v) = \sqrt{u^2 + v^2}$

Notice that the  $u, v$  here are represented by  $i - cx$  and  $j - cy$ .

```
def ideal(i, j, cx, cy, hp, D0):
    D = math.sqrt((i - cx)**2 + (j - cy)**2)
    if hp:
        return 0 if D <= D0 else 1
    return 1 if D <= D0 else 0
```

```
def IdealFilter(n_row, n_col, cutoff_freq, highPass=True):
    center_x = int(n_row/2) + 1 if n_row % 2 == 1 else int(n_row/2)
    center_y = int(n_col/2) + 1 if n_col % 2 == 1 else int(n_col/2)
    return np.array([[ideal(i, j, center_x, center_y, highPass,
        cutoff_freq) for j in range(n_col)] for i in range(n_row)])
```

## 2. Guassian filter

- High-pass  $H(u, v) = 1 - e^{-D^2(u, v)/2D_0^2}$   
where  $D_0$  is cutoff frequency,  $D^2(u, v) = u^2 + v^2$
- Low-pass  $H(u, v) = e^{-D^2(u, v)/2D_0^2}$   
where  $D_0$  is cutoff frequency,  $D^2(u, v) = u^2 + v^2$

Notice that the  $u, v$  here are represented by  $i - cx$  and  $j - cy$ .

```
def gaussian(i, j, cx, cy, hp, sigma):
    coeff = math.exp(-1.0 * ((i - cx)**2 + (j - cy)**2) / (2 * sigma**2))
    if hp:
        return 1 - coeff
    return coeff
```

```
def GaussianFilter(n_row, n_col, sigma, highPass=True):
    center_x = int(n_row/2) + 1 if n_row % 2 == 1 else int(n_row/2)
    center_y = int(n_col/2) + 1 if n_col % 2 == 1 else int(n_col/2)
    return np.array([[gaussian(i, j, center_x, center_y, highPass, sigma)
                      for j in range(n_col)] for i in range(n_row)])
```

For Fourier transformation, the procedure are as follow:

1. Compute Fourier transformation of input image  $F(u, v)$  and shift input image to center the transform

```
shiftedDFT = fftshift(fft2(image))
```

Here `fft2(img)` applies the 2D Fast Fourier Transform to the image `img`. The FFT transforms the image from the spatial domain (where pixel intensities are given) to the frequency domain, showing the frequency components that make up the image. And `fftshift()` shifts the zero-frequency component (the DC component) to the center of the spectrum.

2. Multiply  $F(u, v)$  by a filter function  $H(u, v)$

```
filteredDFT = shiftedDFT * \
    IdealFilter(
        image.shape[0], image.shape[1], cutoff_freq, highPass=isHigh)
```

3. Compute the inverse Fourier transformation of  $H \cdot F$  to obtain the image

```
res = ifft2(ifftshift(filteredDFT))
```

`ifftshift(filteredDFT)` shifts the frequency components back to their original arrangement (with the zero-frequency component at the corners) after they had been previously centered using `fftshift`. `ifft2()` is the 2D Inverse Fast Fourier Transform (IFFT), which applied to convert the filtered frequency domain representation (stored in `filteredDFT`) back into the spatial domain. This will reconstruct the image or signal from its frequency components after filtering was applied in the frequency domain.

Above procedures will be like:

```
def filter_I(image, cutoff_freq, isHigh):
    shiftedDFT = fftshift(fft2(image))
    filteredDFT = shiftedDFT * \
        IdealFilter(
            image.shape[0], image.shape[1], cutoff_freq, highPass=isHigh)
    res = ifft2(ifftshift(filteredDFT))
    return np.real(res)
```

For combining two images, we simply add the high-pass image's and low-pass image's value:

```
def hybrid_img_I(high_img, low_img, cut_h, cut_l):
    res = filter_I(high_img, cut_h, isHigh=True) + \
        filter_I(low_img, cut_l, isHigh=False)
    return res
```

## 2. Image Pyramid

The goal of the image pyramid task is to create multiple representations of an image at progressively lower resolutions. This is achieved by repeatedly applying a smoothing filter to the image and downsampling it to form a pyramid of images. Here's a step-by-step breakdown of the procedure:

1. Initialization A list `pyramid_images` is initialized to store the different levels of the pyramid. The original image is added as the first level. Similarly, the 2D Fast Fourier Transform (FFT) of the image is computed and stored in the `fft_images` list for frequency domain analysis.

```
pyramid_images = [img]
fft_images = [np.fft.fft2(img)]
```

2. Iterative Pyramid Construction The process of building the pyramid involves iteratively applying a Gaussian filter to the image, followed by downsampling. For each level of the pyramid:
  - Gaussian Smoothing : A Gaussian filter is applied to the current image to remove high-frequency content. This smoothing operation helps prepare the image for downsampling, reducing aliasing artifacts.
  - Downsampling : The smoothed image is downsampled by a factor of 2 in both the x and y directions. This reduces the image size by half at each level of the pyramid.

- The process continues until reach `num_levels` . Each downscaled image is added to the `gaussian_images` list .

```
def gaussian_pyramid(image, levels):  
    gaussian_images = [image]  
    cutoff_frequency = 10  
  
    for i in range(1, levels):  
        smoothed = filter(image, cutoff_frequency, low_pass=True,  
filter_type='gaussian')  
  
        downsampled = smoothed[::2, ::2]  
        gaussian_images.append(downsampled)  
  
        image = downsampled  
    return gaussian_images
```

### 3. Colorizing the Russian Empire

In this task, the goal is to automatically colorize Prokudin-Gorskii's glass plate images. These images consist of three grayscale channels corresponding to the red, green, and blue filters. To produce the final color image, these three channels need to be accurately aligned and merged into an RGB image.

To achieve this, we implemented four different alignment methods: Direct Addition, MSE Alignment, Canny Edge Detection Alignment, and Gaussian Pyramid Alignment. Each method uses a distinct strategy for alignment, and we will explain each method in detail below. Additionally, Gaussian Pyramid Alignment is the most efficient method based on the time taken. It builds a multi-level pyramid and performs alignment on the low-resolution image first, finding the appropriate offset. This significantly reduces unnecessary computations in the high-resolution image.

#### Find Alignment offset

This function ( `align_by_translation` ) is essential for finding the best offset to align one image with another. It is used in various alignment methods such as MSE Align, Edge Align, and Pyramid Align. Here's how the function works: This function aligns one image (`image_to_align`) with a reference image (`reference`) by translating it within a specified range (`max_offset`). It searches for the best offset in both x and y directions that minimizes the mean squared error (MSE) between the translated image and the reference.

```
def align_by_translation(reference, image_to_align, max_offset=20):
    best_offset = (0, 0)
    min_error = float('inf')

    for y_offset in range(-max_offset, max_offset + 1):
        for x_offset in range(-max_offset, max_offset + 1):
            translated_image = np.roll(image_to_align, shift=(y_offset,
x_offset), axis=(0, 1))

            # Calculate the mean square error between the translated image and
the reference image
            error = mse(reference, translated_image)

            # Find the location of minimum error
            if error < min_error:
                min_error = error
                best_offset = (y_offset, x_offset)
```

## 1. Direct Addition Method

This is the simplest method. It directly stacks the three channels, assigning the top, middle, and bottom parts of the image to the blue, green, and red channels respectively, and then merges them into one color image. While this method is very fast (with the shortest computation time), it often results in significant color misalignment due to the lack of any alignment adjustments.

```
# ===== Direct Add =====
def direct_add(image):
    h, w = image.shape

    div = h//3
    size = image[:div, :]

    blue = np.zeros((size.shape[0], size.shape[1], 3), dtype=np.uint8)
    green = np.zeros((size.shape[0], size.shape[1], 3), dtype=np.uint8)
    red = np.zeros((size.shape[0], size.shape[1], 3), dtype=np.uint8)

    blue[:, :, 0] = image[:div, :]
    green[:, :, 1] = image[div:2*div, :]
    red[:, :, 2] = image[2*div:3*div, :]

    output = cv2.add(red, green) # add red and green
    output = cv2.add(output, blue) # add blue
    return output
```

## 2. MSE Alignment Method

To address the misalignment issue, the MSE alignment method uses mean squared error (MSE) to adjust the green and red channels by shifting them, aiming to minimize the pixel-wise error with the blue channel. This method improves alignment accuracy but requires searching through multiple offset values to find the optimal alignment, making it computationally expensive and slower to execute.

```
# ===== MSE Align =====
def mse_align(channel, ref_channel="blue"):
    blue_channel, green_channel, red_channel = channel

    ref_channel = blue_channel if ref_channel == "blue" else green_channel
    if ref_channel == "green" else red_channel
    offsets = [(0, 0), (0, 0), (0, 0)]
    for i, align_channel in enumerate([blue_channel, green_channel,
red_channel]):
        if align_channel is ref_channel:
            continue
        _, offset = align_by_translation(ref_channel, align_channel)
        offsets[i] = offset
    color_image = np.dstack((np.roll(blue_channel, shift=offsets[0], axis=
(0, 1)),
                                np.roll(green_channel, shift=offsets[1], axis=(0,
1)),
                                np.roll(red_channel, shift=offsets[2], axis=(0,
1)),
                                ))
    return color_image
```

## 3. Canny Edge Detection Alignment Method

This method applies Canny edge detection to each channel to extract the edges, then aligns the green and red channels based on the edges, minimizing the MSE between the aligned edges and the reference (blue) edges. This method is particularly effective for images with strong structural edges and is slightly faster than the MSE method, as it focuses on aligning the edges instead of raw pixel values.

```
# ===== Edge Align =====
def canny_edge_align(channel, ref_channel="blue"):
    blue_channel, green_channel, red_channel = channel

    blue_edges = edge_detection(blue_channel)
    green_edges = edge_detection(green_channel)
    red_edges = edge_detection(red_channel)

    ref_edges = blue_edges if ref_channel == "blue" else green_edges if
ref_channel == "green" else red_edges
    offsets = [(0, 0), (0, 0), (0, 0)]
    for i, align_edges in enumerate([blue_edges, green_edges, red_edges]):
        if align_edges is ref_edges:
            continue
        _, offset = align_by_translation(ref_edges, align_edges)
        offsets[i] = offset
    color_image = np.dstack((np.roll(blue_channel, shift=offsets[0], axis=
(0, 1)),
                                np.roll(green_channel, shift=offsets[1], axis=
(0, 1)),
                                np.roll(red_channel, shift=offsets[2], axis=
(0, 1)),
                                ))
    return color_image
```

#### 4. Gaussian Pyramid Alignment Method (Our final method)

Finally, we implemented the alignment method based on the Gaussian pyramid, which is also our final method. This approach utilizes the technique from the second task, Image Pyramid. It constructs a Gaussian pyramid by progressively downscaling the image, performs initial alignment at the lowest level, and then applies the alignment results to higher levels. By aligning at smaller scales, the search space is reduced, resulting in higher computational efficiency compared to the previous methods.



```
# ===== Pyramid Align =====
def gaussian_pyramid_align(image, channel, level=10, scaling_factor=0.75,
max_offset=20, ref_channel="blue"):
    pyramid_image = generate_image_pyramid(image, levels=level,
scaling_factor=scaling_factor)
    height = pyramid_image.shape[0] // 3
    blue_pyramid = pyramid_image[0:height]
    green_pyramid = pyramid_image[height:2*height]
    red_pyramid = pyramid_image[2*height:3*height]

    ref_pyramid = blue_pyramid if ref_channel == "blue" else green_pyramid
    if ref_channel == "green" else red_pyramid
    offsets = [(0, 0), (0, 0), (0, 0)]
    scaling_total = image.shape[0] / pyramid_image.shape[0]
    for i, align_channel in enumerate([blue_pyramid, green_pyramid,
red_pyramid]):
        if align_channel is ref_pyramid:
            continue
        _, offset = align_by_translation(ref_pyramid, align_channel,
max_offset=max_offset)
        offset = (offset[0] * scaling_total, offset[1] * scaling_total)
        offsets[i] = offset

    color_image = np.dstack((np.roll(channel[0], shift=offsets[0], axis=(0,
1)),
                                np.roll(channel[1], shift=offsets[1], axis=(0,
1)),
                                np.roll(channel[2], shift=offsets[2], axis=(0,
1)),
                                ))

    return color_image
```

## 4. Execution method

The default images are images from the `my_data` folder. To change the images, you need to modify the image path.

### Task 1 - Hybrid image

```
python task1_hybrid_image.py
```

### Task 2 - Image Pyramid

```
python task2_pyramid_image.py
```

### Task 3 - Colorizing the Russian Empire

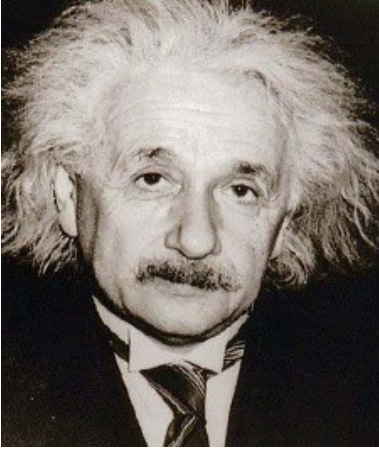
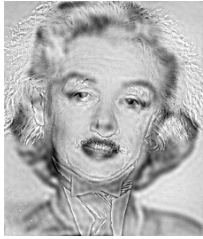


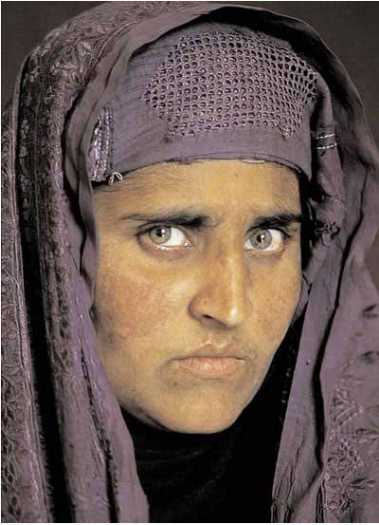


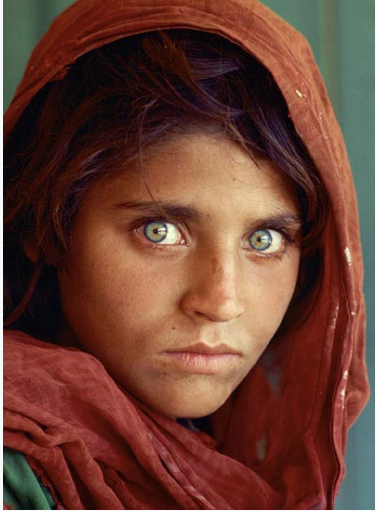




```
python task3_coloring.py
```

## Experimental Results





---

### 1. Hybrid image

In Task 1, Hybrid Image, we implemented ideal and Gaussian filters to extract and combine high and low-frequency information. The results in the figure show that while the ideal filter enhances edge sharpness slightly more, the Gaussian filter provides smoother transitions. However, the overall differences between the two filters are minimal, as both effectively blend high and low-frequency components, resulting in comparable hybrid images.

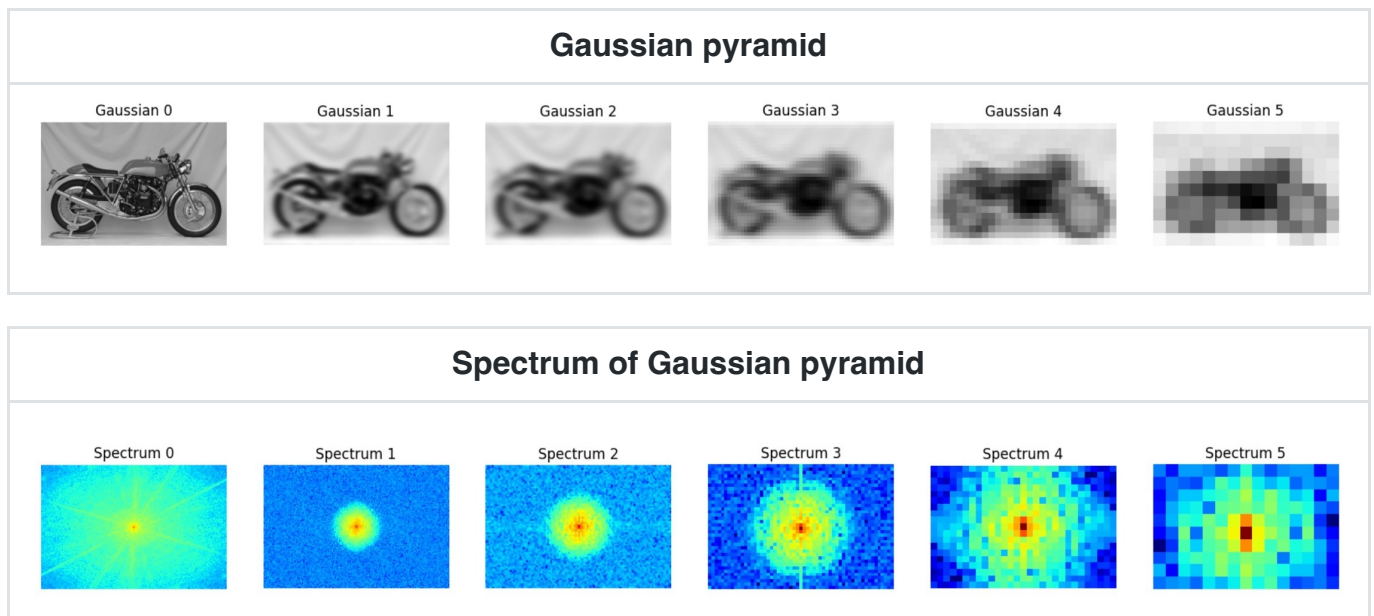
Input (High)	Ideal	Gaussian	Input (Low)
			
			
			

We also attempted using other photos from the internet (placed in the mydata folder), and the results are as follows:

Input (High)	Ideal	Gaussian	Input (Low)
			

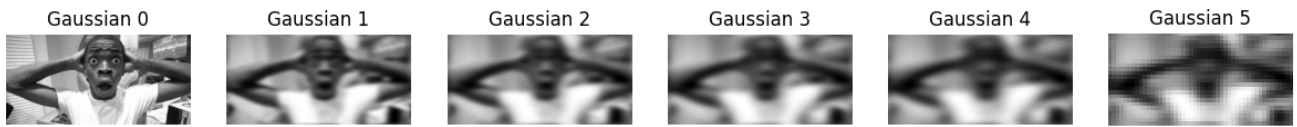
## 2. Image Pyramid

In Task 2, we used a Gaussian filter to create an image pyramid. As we increase the levels, the images become progressively blurrier. From the frequency spectra of the Gaussian pyramid, we can observe that with each level, the center of the spectrum, representing low-frequency components, becomes more dominant. This indicates that as we blur and downsample the image, the low-frequency parts take over. From Spectrum 0 to Spectrum 5, we can see the high-frequency details gradually fading, causing the edges and finer details to become less noticeable, as the high-frequency components are weakened with each blurring step.

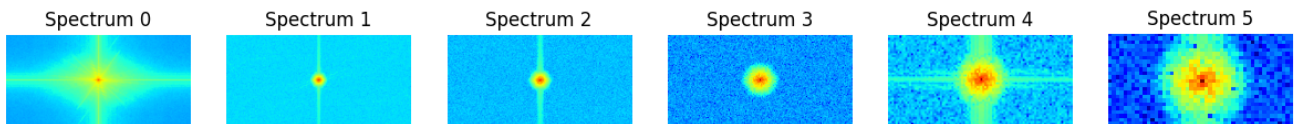


Additionally, we applied the Gaussian image pyramid method to images sourced from the web, and the results are shown in the figure below. These demonstrate the same pattern of increasing dominance of low-frequency components as we move through the pyramid levels.

### Gaussian pyramid (my\_data)



### Spectrum of Gaussian pyramid (my\_data)



## 3. Colorizing the Russian Empire

In the "Colorizing the Russian Empire" task, we took the idea from task 2 and used a Gaussian pyramid in task 3 to speed up the alignment process. After a lot of testing, we found that setting the pyramid to 10 levels, using a scale factor of 0.75, and applying a Gaussian high-pass filter at level 5 to catch high-frequency features (like edges) gave us the best alignment results across different images. This method also worked really efficiently. The table below shows the results of combining multiple images, and you can see that most of them are well-aligned and colorized nicely.

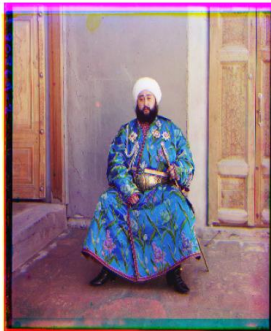


## The results of colorizing the Russian Empire (Gaussian pyramid method)

workshop



emir



monastery



three\_generations



melons



onion\_church



train



tobolsk



icon



nativity



cathedral



village



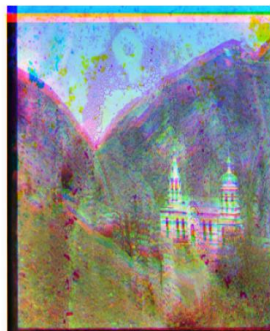
00125v





lady



00056v



We also sourced additional images from the website [CMU 15463 Project](#) to perform colorizing. As shown in the table below, the images colorized using the Gaussian pyramid method are very close to the ground truth (GT) provided on the website. This demonstrates that our approach effectively reproduces accurate colors and fine details, making the results highly comparable to the reference images.

Colorizing the Russian Empire	from the website (CMU 15463 Project)
	
Our gaussian pyramid method	Ground truth

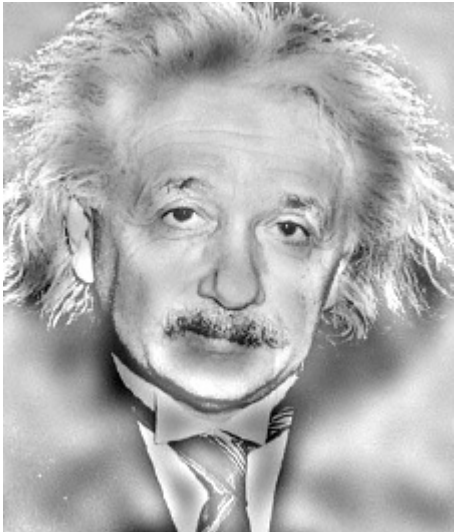
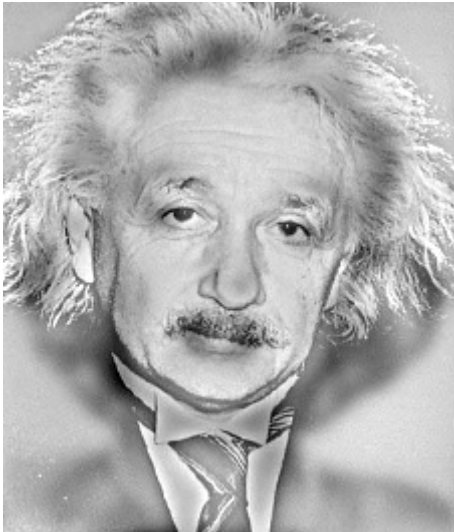
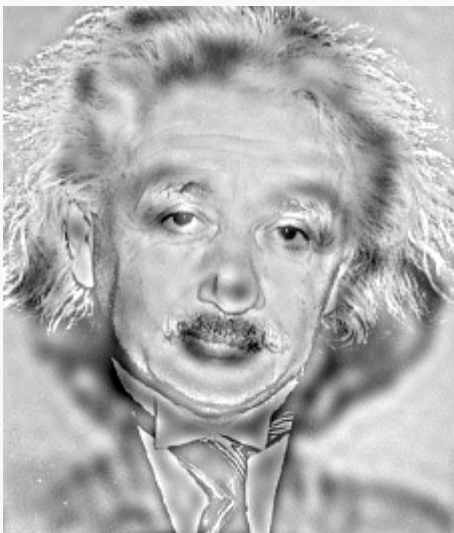
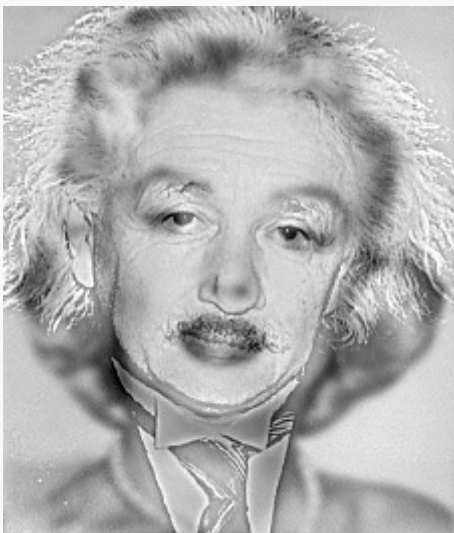


## Discussion

### Hybrid image




In Task 1, we experimented with different cutoff frequencies and found that the smaller the cutoff frequency, the resulting hybrid image resembled the high-frequency image (Einstein) more. Conversely, when the cutoff frequency was larger, the hybrid image leaned more towards the low-frequency image (Marilyn).

This is because the cutoff frequency determines how much of the high and low-frequency components are retained in the final image. A smaller cutoff frequency allows more high-frequency details to pass through, emphasizing the sharper, finer details from the high-frequency image. On the other hand, a larger cutoff frequency permits more low-frequency information, which preserves the broad, smooth features from the low-frequency image. As a result, adjusting the cutoff frequency effectively controls the balance between the two images in the hybrid, shifting focus between the high-frequency details and the low-frequency structure.



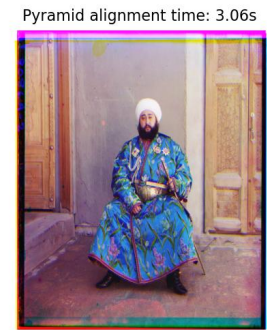
Cutoff frequency	Ideal	Gaussian
10		
20		
30		



Cutoff frequency	Ideal	Gaussian
40		
50		

## Colorizing the Russian Empire

In addition to the Gaussian pyramid, we implemented three other methods: "Direct Addition," "MSE Align," and "Canny Edge Align," and compared all four approaches. When using larger images, such as *emir.tif*, we observed that the MSE and Canny Edge methods took around 80 seconds to complete the colorization, while the Gaussian pyramid method finished in just 2 to 3 seconds. As shown in the image below, this method not only significantly improved efficiency but also delivered more precise results. We believe this is due to the high-pass filter, which effectively captures edge information, leading to better alignment.



## Conclusion

---

In this assignment, we explored several image processing techniques, including hybrid images, image pyramids, and the colorization of historical images. Each method required a combination of filtering, downsampling, and image alignment to achieve the desired results. The task of colorizing Prokudin-Gorskii's glass plate negatives was particularly challenging due to the need for precise alignment of the red, green, and blue channels. By implementing four different alignment methods — Direct Addition, MSE Align, Edge Align, and Gaussian Pyramid Align — we demonstrated how different strategies can affect both the quality and efficiency of image alignment. Among these, the Gaussian Pyramid Alignment proved to be the most efficient, as it allowed us to perform alignment at lower resolutions and refine it progressively, greatly reducing computational overhead.

Through this assignment, we gained valuable insights into the power of multi-scale image analysis and the importance of efficient algorithms in processing high-resolution images. These techniques not only illustrate fundamental principles of image processing but also have broad applications in modern computer vision.

## Work Assignment Plan

---

- 110705017 何翊華
  - Hybrid Image Task : implement the hybrid image method, which included designing the high-pass and low-pass filters
  - Image Pyramid Task : implement the image pyramid construction using Gaussian filters and downsampling
  - Code Optimization and Testing
  - Report Writing and Documentation
- 313551097 鄭淮薰

- Image Pyramid Task : implement the image pyramid construction using Gaussian filters and downsampling
  - Colorizing the Russian Empire Task : implement the four alignment methods: Direct Addition, MSE Alignment, Edge Alignment, and Gaussian Pyramid Alignment.
  - Code Optimization and Testing
  - Report Writing and Documentation
- 313551098 張懷齡
    - Hybrid Image Task : implement the hybrid image method, which included designing the high-pass and low-pass filters
    - Colorizing the Russian Empire Task : implement the four alignment methods: Direct Addition, MSE Alignment, Edge Alignment, and Gaussian Pyramid Alignment.
    - Code Optimization and Testing
    - Report Writing and Documentation