CV HW2 Group 19 Report

0756079 陳冠聞, 0756138 黃伯凱

April 7, 2019

1 Introduction

In this report, we will show results of three tasks respectively. First, hybrid images, we will hybrid low-passed image and high-passed image. It is interesting to find out the result will be distinguished into two distinct images if we look at it at different distance. Secondly, image pyramid, we will show both gaussian pyramid and laplacian pyramid of each image in spatial domain and frequency domain. Last, colorizing the russian empire, we align and combine three channels (BGR) of a single image and restore it into a colorful image.

We implement these tasks using python3, and only import NumPy for matrix operation and OpenCV for image I/O, all the other functions are implemented by ourselves.

2 Hybrid Image

We hybrid 6 sets of images (12 images) and exchange the high-pass and loss-pass image in each set. That is, there are 12 results and they will be in the "res" directory.

2.1 Implementation

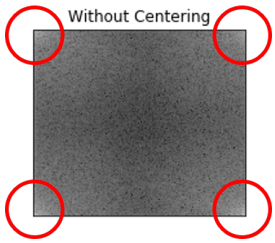

Figure 1: Frequency Image without Centering

In order to hybrid two images, we have to sum a low-pass image and a high-pass image. Since frequency is what we desire to filter, we must get the frequency image of the input. Notice that the zero frequency of the frequency image is at the four corners we circled in Figure 1. We have to shift the zero frequency to the center of the image to enable us not only look at it more friendly but also easier to apply a filter such as Figure 2(c).

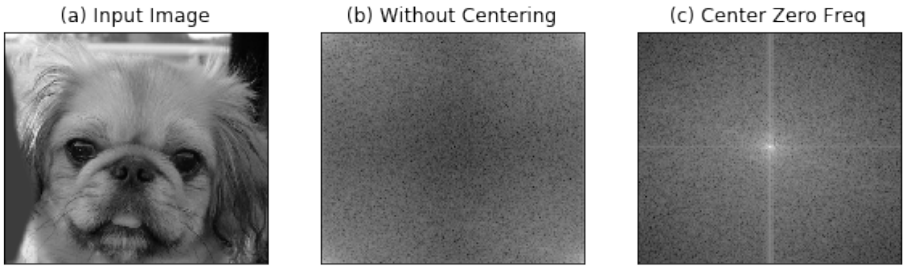

Figure 2: Centering zero frequency

Assume the axis of the image in spatial domain is x,y while u,v in frequency domain, and the size of the image is M. Since the image size of frequency domain is the same as the size of spatial domain, we have to shift the frequency image $\frac{M}{2}$ pixels to the right and $\frac{N}{2}$ pixels to the up to center the zero frequency like Figure 2(c). However, it is difficult to calculate the image shifting in frequency domain. So we will do the inverse Fourier transform of the shifted frequency image, and complete the calculation in spatial domain in order to simplify the computation.

$$\begin{aligned}
\mathscr{F}^{-1}\{F(u-\frac{M}{2},v-\frac{N}{2})\} &= \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u-\frac{M}{2},v-\frac{N}{2})e^{j2\pi(ux+vy)}dudv \\
&= \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u-\frac{M}{2},v-\frac{N}{2})e^{j2\pi[(u-\frac{M}{2})x+(v-\frac{N}{2})y]}e^{j2\pi(\frac{M}{2}x+\frac{N}{2})y}dudv \\
&= f(x,y)e^{j\pi(Mx+Ny)} \\
&= f(x,y)e^{jMx\pi}e^{jNy\pi} \\
&= f(x,y)[cos(Mx\pi)+jsin(Mx\pi)][cos(Ny\pi)+jsin(Ny\pi)] \\
&= f(x,y)(-1)^x(-1)^y \\
&= f(x,y)(-1)^{x+y}
\end{aligned} \tag{1}$$

As Eq.(1) shows, an alternative way to shift the image in frequency domain is to multiply the spatial image by $(-1)^{x+y}$. That is, we have to shift the image in spatial domain before we Fourier transform it. Figure 3 shows how we center the zero frequency.

```python
# Shift the image in Spatial domain
def shiftImg(img, r, c):
    for r_ in range(r):
        for c_ in range(c):
            if (r_+c_)%2 != 0:
                img[r_, c_] = -img[r_, c_]
    return img
```

Figure 3: Code Snippet for Centering zero frequency

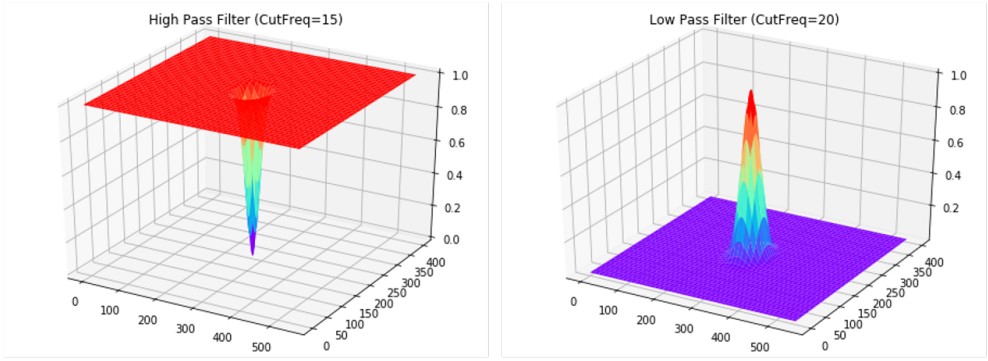

Figure 4: Gaussian Filter

After shifting the image, we are able to apply high-pass and low-pass filter to the image. What we have to do next is to Fourier transform the shifted image and simply multiply it by the filter in Figure 4. Take high-pass filter for example, the low frequency part is obviously smaller after filtering (Figure 5). Then, we can sum the filtered images to get the hybrid image in frequency domain.

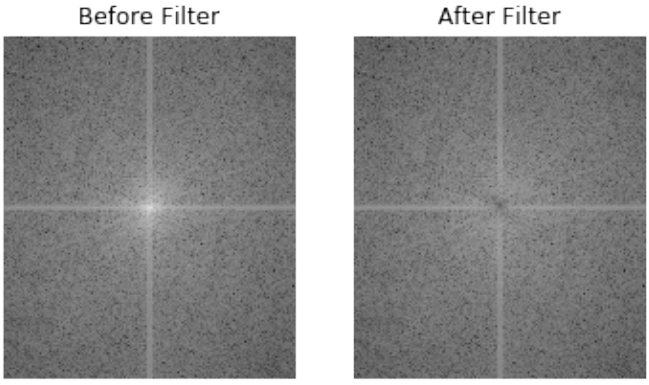

Figure 5: Comparison before and after filtering

```python
def gaussian(u, v, sigma):
    return np.exp(-(np.square(u) + np.square(v))/(2*(sigma**2)))

# Calculate and plot the gaussian filter
def gaussian_filter(r, c, cut_freq, lowPass):
    center_x = int(r/2)
    center_y = int(c/2)

    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(111, projection='3d')

    X = np.arange(r)
    Y = np.arange(c)
    Y, X = np.meshgrid(Y, X)

    if lowPass:
        ax.set_title('Low Pass Filter (CutFreq=%s)' %(cut_freq))
        Z = gaussian(X-center_x, Y-center_y, cut_freq)
    else:
        ax.set_title('High Pass Filter (CutFreq=%s)' %(cut_freq))
        Z = 1 - gaussian(X-center_x, Y-center_y, cut_freq)

    surface = ax.plot_surface(X, Y, Z, cmap=plt.get_cmap('rainbow'))
    plt.show()
    return Z
```

Figure 6: Code Snippet for High/Low Pass Filter

Finally, inverse Fourier transform the frequency image and multiply $(-1)^{x+y}$ again to shift the spatial image back. What we obtain is the result of hybrid image in spatial domain.

```python
# Filter the DFT image and inverse it to spatial domain
def filteredImage(img, r, c, filter):
    imgDFT = fft2(shiftImg(img, r, c))
    tmp = np.multiply(imgDFT, filter)
    result = np.real(ifft2(tmp))
    result = shiftImg(result, r, c)
    return result
```

Figure 7: Code Snippet for filtering an image

2.2 Result

We show only the hybrid image of high-pass bicycle and low-pass motorcycle in this section, the rest of them are in the "res" directory. Figure 8(a) is our filtered image while Figure 8(b) is ideal filtered image, and Figure 8(c) is the comparison between ours and ideal result. It is obvious that the result of ideal filter has some noise in the background. This is because of the **Ringing Effect**. We will talk about this effect in discussion.

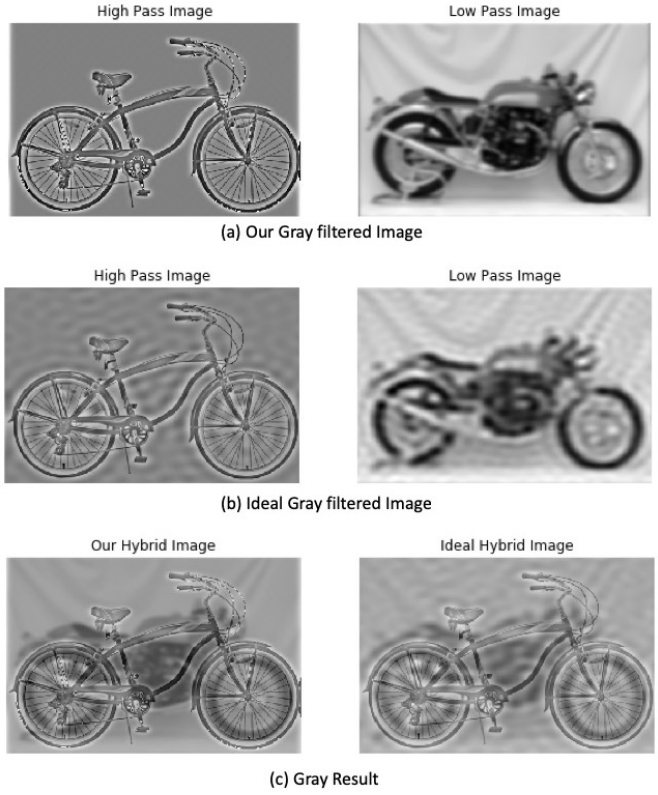

Figure 8: Gray Scale Hybrid Image

We also implement the RGB hybrid image. It is similar to gray scale hybrid image. The only difference is that we have to filter each channel respectively and merge them all at the end.

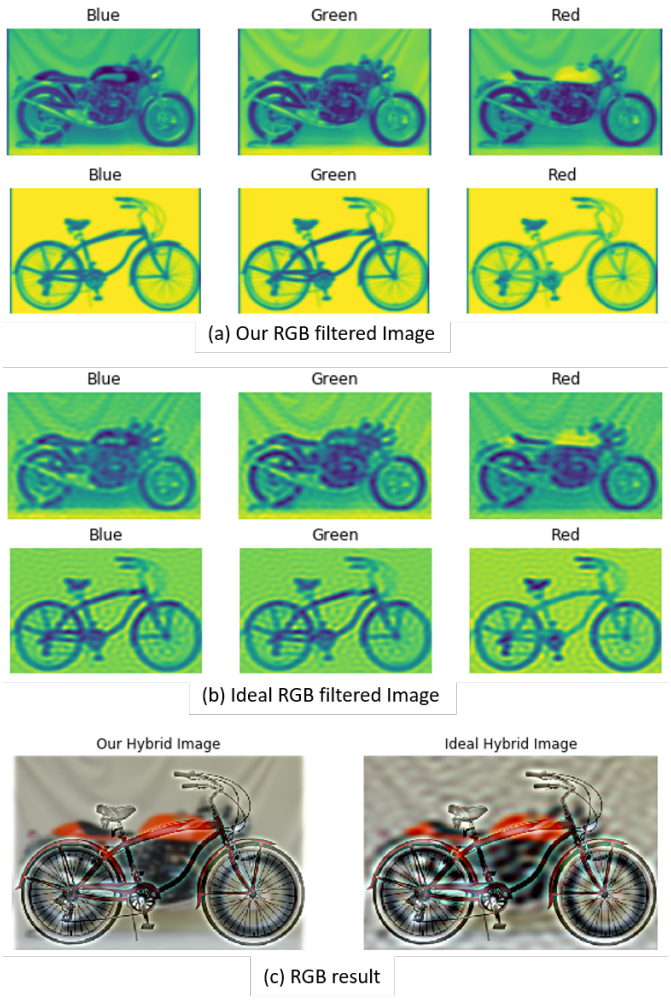

Figure 9: RGB Hybrid Image

3 Image Pyramid

To deal with multi-scale problem, image pyramid is a widely used approach. There are two common pyramid, which are Gaussian pyramid and Laplacian pyramid, so we will implement these two pyramid and show the result when image are applied to these two pyramid.

3.1 Implementation

We first implement the 2-D convolution and Gaussian filter. For the Gaussian kernel, we simply compute the value of probability density function of a 2-D Gaussian distribution (the mean is at the center of the kernel) at each position of the kernel, and normalize the kernel to make sure the sum of kernel is exactly 1. For the 2-D convolution, we use cross-correlation with flipped kernel instead, because it's equivalent to convolution and easy to implement. In addition, to make the output shape fixed after convolution, we use zero-padding approach. The implementation details of Gaussian filter and 2-D convolution are shown in Figure 10 and Figure 11.

```
def gaussian_pdf_2D(x, y, sigma=1):
    exp_term = -(x*x + y*y) / (2.*sigma*sigma)
    const_term = 1/(2*pi*(sigma**2))
    return const_term * exp(exp_term)

def get_gaussian_filter(kernel_size=(3,3), sigma=1):
    gaussian_kernel = np.zeros((m,n))
    center_x = (n+1) // 2 -1
    center_y = (m+1) // 2 -1
    for u in range(n):
        for v in range(m):
            gaussian_kernel[v, u] = gaussian_pdf_2D(u-center_x, v-center_y, sigma)
    gaussian_kernel /= gaussian_kernel.sum()
    return gaussian_kernel
```

Figure 10: Code Snippet for Gaussian kernel

```python
def zero_padding(img, size=(1,1)):
    m, n = img.shape
    img_pad = np.zeros((m+2*oy, n+2*ox))
    img_pad[oy:-oy, ox:-ox] = img
    return img_pad

def flip_kernel(kernel):
    return kernel[::-1, ::-1]

def convolution_2D(img, kernel, padding=True):
    kernel = flip_kernel(kernel)
    p, q = kernel.shape
    ox = (q-1)//2
    oy = (p-1)//2
    if padding:
        img = zero_padding(img, size=(oy,ox))
        m, n = img.shape
    output_img = np.zeros((m, n))
    for cy in range(oy, m-oy):
        for cx in range(ox, n-ox):
            img_window = img[(cy-oy):(cy+oy)+1:, (cx-ox):(cx+ox)+1]
            output_img[cy, cx] = np.sum(img_window*kernel)
    output_img = output_img[oy:-oy, ox:-ox]
    return output_img
```

Figure 11: Code Snippet for 2D Convolution

Second, we build the Gaussian pyramid. The first layer (finest layer) is just the input image, and the following layers is apply Gaussian filter to the previous layer then do subsampling, and all the way down do the last layer (coarsest layer).

Finally, we build the Laplacian pyramid. The n-th layer of Laplacian pyramid L_n is n-th layer of Gaussian pyramid G_n subtract from upsampled result of (n+1)-th layer of Gaussian pyramid, which can be written as $L_n = G_n - upsample(G_{n+1})$.

```python
def image_pyramid(img, gaussian_filter, num_layers=5, ratio=2):

    # Gaussian Pyramid
    img_layer = img.copy()
    gaussian_pyramid = [img_layer]
    for i in range(num_layers):
        img_smooth = convolution_2D(img_layer, gaussian_filter)
        m, n = img_layer.shape
        img_layer = subsampling(img_smooth, size=(int(m/ratio), int(n/ratio)))
        gaussian_pyramid.append(img_layer)

    # Laplacian Pyramid
    laplacian_pyramid = []
    for i in range(1, len(gaussian_pyramid)):
        img_high = gaussian_pyramid[i-1]
        img_low = gaussian_pyramid[i]
        img_up = nearest_interpolation(img_low, img_high.shape)
        laplacian = img_high - img_up
        laplacian_pyramid.append(laplacian)

    return gaussian_pyramid, laplacian_pyramid
```

Figure 12: Code Snippet for Image Pyramid

3.2 Result

We show the image pyramid result of *"Afghan girl before.jpg"* in the Figure 13. In this image pyramid, there are 5 layers in total, and we choose $\sigma = 1.5$

for the Gaussian kernel with kenrel size 7×7, also, downsample ratio is set to 2 for all layers. In the Figure 13, the row from top to down represents the Gaussian pyramid image, Laplacian pyramid, Gaussian pyramid magnitude spectrum and Laplacian pyramid magnitude spectrum respectively. And the column from left to right represents the finest layer to the coarsest layer.

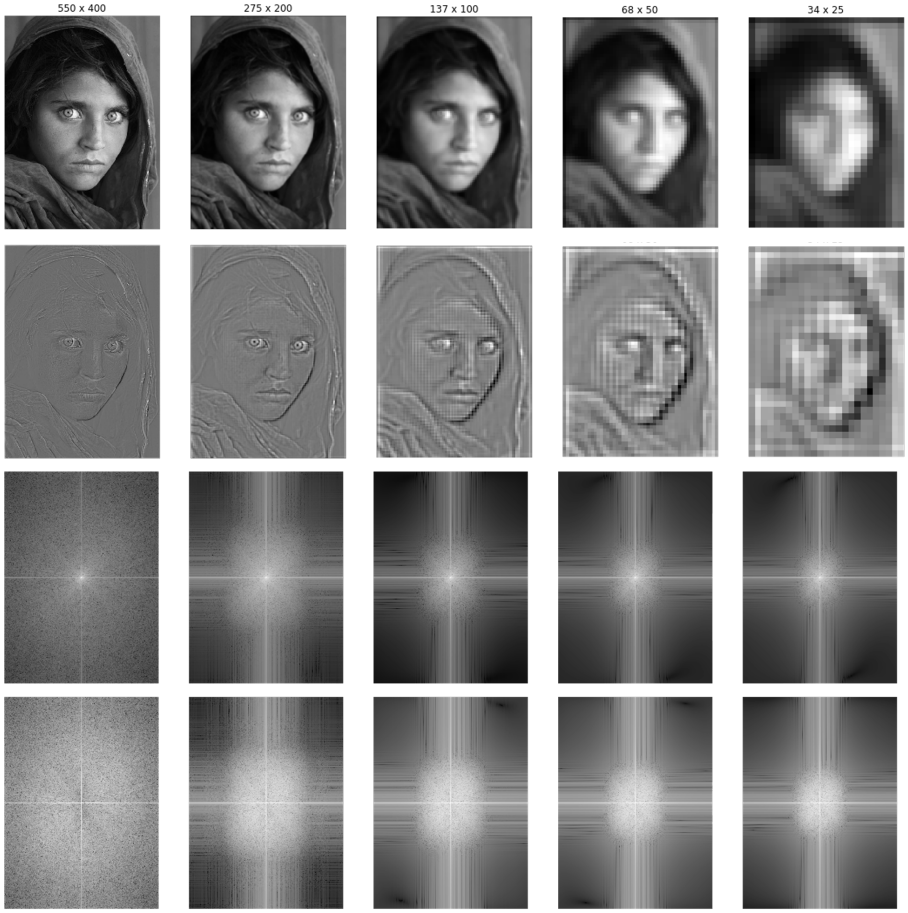

Figure 13: Result of Image Pyramid

4 Colorizing the Russian Empire

4.1 Implementation

To produce a color image from the digitized ProkudinGorskii glass plate images with as few visual artifacts as possible, we need to have different color channels of image, and align each channel of the image together.

We first divided origin image into three equal size parts, which correspond to blue, green and red channel respectively. For the alignment, it's seem that we can directly use sum of square difference (SSD) to measure the alignment error between two different channels, and use sliding window approach to find the best alignment, but in fact, the result is not good enough.

The reason is that the brightness of the same pixel in each channel might be different, so the lowest error may not be the best alignment. In addition, the boundary of each channel is very noisy and contain less useful information to do alignment. So we decide to crop the boundary (1/10 of image size) and apply the sobel filter to get the edge image of each channel first, then use sliding window to find the lowest SSD between different channel edge. The procedure of our algorithm is shown in Figure 14.

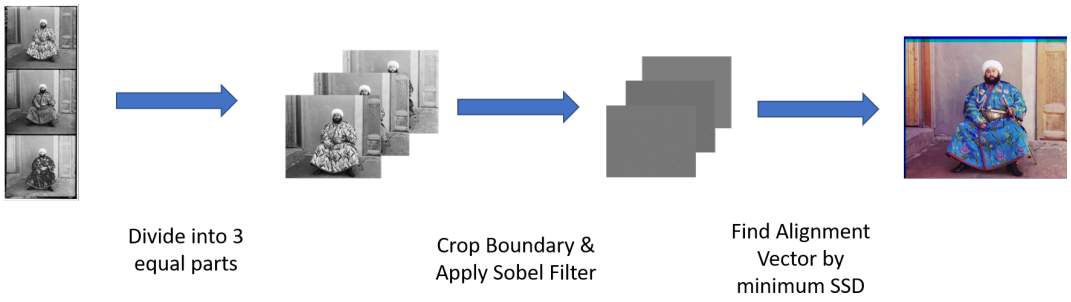

Figure 14: Colorizing Algorithm Procedure

Furthermore, for different resolution of image, we use different strategies to find the minimum SSD. For low-resolution images ($\approx 400 \times 1000$), we use window size of $[-50, 50]$ to do brute-force search. For high-resolution images ($\approx 3700 \times 9600$), we use multi-scale pyramid approach, which is searching minimum SSD in larger windows size for downsampled images,

then refine the alignment vector in progressively larger images. The reason to use multi-scale pyramid approach is because it's more computationally efficient compared to directly search large window in high resolution image.

4.2 Result

In our algorithm, the default setting for multi-scale pyramid is to downsample to $\frac{1}{16}$ of origin image size, and upscale by 2 in each refine iteration. The default initial window size is the height of downsampled image divided by 5, and shrink window size by 2 at each iteration. Nevertheless, we change this setting for some images to get better performance. The produced color images are shown below (the images have been manually cropped to remove the weird boundary pattern). Full resolution color images can be found in in the "res" directory.

Figure 15: All produced color images

5 Discussion

5.1 Ringing Effect

Ringing effect happens if we use ideal high/low pass filter, it is caused by distortion or loss of high frequency information in image. The term "ringing" is because the output signal oscillates at a fading rate around a sharp transition in the input. It will results as spurious signals near sharp transitions in a signal.

It is more intuitive to explain it in math. Ringing effect happens if we multiply the input frequency image by an ideal filter, which is a 2D square wave. Since the inverse fourier transform of square wave is a sinc wave (Figure 16).

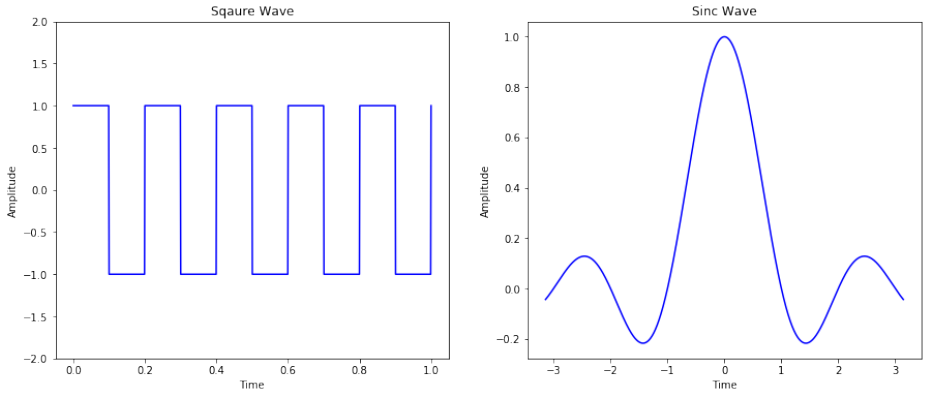

Figure 16: Square Wave and Sinc Wave

In addition, due to the convolution theorem (Eq.(2)), multiplying a frequency image by ideal filter is equivalent to convolution a spatial image with a sinc wave like Fig(17).

$$\mathscr{F}^{-1}\{F_1(\omega) \cdot F_2(\omega)\} = f_1(x) * f_2(x) \tag{2}$$

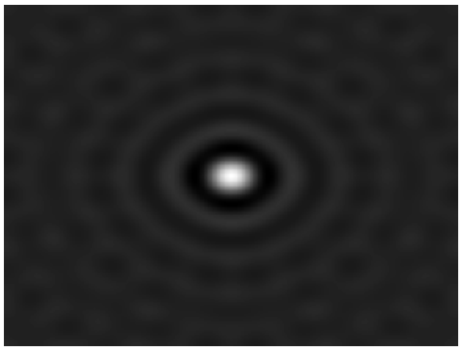

Figure 17: Sinc Filter

As a result, there will be some noise near sharp transition in spatial domain.

5.2 Pyramid Approach

In the lecture, we are taught that image pyramid is used to deal with multi-scale problem, such as template matching with different scale. In *Colorizing the Russian Empire*, we found that the idea of pyramid can even be used to speed up computation, especially when image size is very large. Take the alignment 3000×3000 image with SSD as example, if we want to search the all possible alignment with brute-force, it will need to exhaust search the 3000×3000 offset. On the other hand, if we downsample the image by 2 all the way to $\frac{1}{16}$ and do exhaust search in the lowest resolution image, then do refinement and shrink window size by 2 all the way to origin image size, in the end, we only need to search about 11×11 offset in the highest resolution, which save a lot of computation time.

6 Conclusion

In the homework, we implement three tasks including *Hybrid Image*, *Image Pyramid* and *Colorizing the Russian Empire*. For the *Hybrid Image*, we implement the high/low pass filter and show the hybrid image. Furthermore, we discuss the *Ringing Effect* which is caused by ideal filter in the discussion part. For the *Image Pyramid*, we implement the Gaussian pyramid and

Laplacian pyramid of the image, and show each layer of pyramid and its corresponding magnitude spectrum. For the *Colorizing the Russian Empire*, we use sobel filter with SSD approach to find the alignment of different color channels, and show the result of all given images. In addition, we discuss the brute-force search approach and pyramid search approach in the discussion part.

7 Work Assignment Plan

0756138 黃伯凱: Hybrid Image and half of the report.

0756079 陳冠聞: Image Pyramid, Colorizing images and half of the report.