DIP report HW4

Problem 1: DIGITAL HALFTONING

A gray-scale image sample1.png and a dither matrix I2 are shown in Fig. 1.(a) and (b), respectively.

(a) (10 pt) According to the dither matrix I2 , please perform dithering to obtain a binary image **result1.png**.

1	2
3	0

(b) Dither Matrix I_2

Ans: 首先先將2x2的dither matrix根據公式轉為threshold

- **Determine the threshold matrix**

$$T(i,j) = 255 \cdot \frac{I(i,j) + 0.5}{N^2}$$

```
[[ 95.625 159.375]
 [223.125  31.875]]
```

將image加上Gaussian noise，再來根據將其對要處理的image長、寬做拼貼，使其每個pixel都有對應的threshold，大於threshold的設為1（灑點），小於為0（不灑點）。
original input:

output:

沒有先add noise

有先add noise

(b) (15 pt) Expand the dither matrix I2 to I256 (256 × 256) and use it to perform dithering. Output the result as **result2.png**. Compare result1.png and result2.png along with some discussions.

Ans: 利用general form 進行dither matrix的擴展，將2x2的dither matrix遞迴擴展至256x2562的dither matrix，再進行與I2同樣的操作。注意的是dither matrix的長寬256無法整除sample image的長寬622，因此在拼貼過程需要多考慮一些邊緣的情況。

- **The general form of the NxN dither matrix**
  - **2x2 → 4x4 → 8x8 → 16x16…**

$$I_{2n}(i,j)=\begin{bmatrix} 4I_n(i,j)+1 & 4I_n(i,j)+2 \\ 4I_n(i,j)+3 & 4I_n(i,j)+0 \end{bmatrix}$$

output:

沒有先add noise

先add noise

result2相較於result1的整體感覺細緻許多，在色彩的呈現也更有對比度，也比較沒有一塊塊pattern拼貼的感覺。推測是因為256x256的dither matrix轉為threshold時的threshold數值分佈較為細膩，因此顏色也會呈現的比較多對比（減少都落在某個區間的情況），並且threshold數值種類較多，不同threshold的分佈面積較大，因此在pattern拼貼上的結果呈現格子也相對不明顯。

dithering前有沒有先add noise，對我來說肉眼看效果不大。

(c) (25 pt) Perform error diffusion with Floyd-Steinberg and Jarvis' patterns on sample1.png. Output the results as **result3.png** and **result4.png**, respectively. You may also try more patterns and show the results in your report. Discuss these patterns based on the results.
You can find some masks here (from lecture slide 06. p21)

Ans:

error diffusion with Floyd-Steinberg:

整體pattern出現的狀況相較dithering又更少了，整張圖由pattern構成的感覺減少，貓咪邊界也處理的比較細緻。

error diffusion with Jarvis:

感覺貓咪的臉的五官似乎對比度變大了，貓咪的邊界似乎也比較清晰，但是點的呈現上感覺沒那麼細緻，部份花色光影反而不見了，而且某種拼貼紋路變得比較明顯。

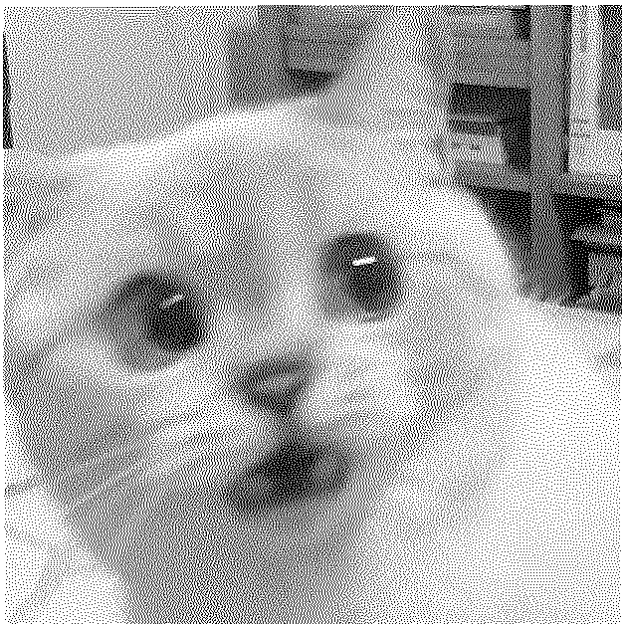
error diffusion + serpentine scanning with Floyd-Steinberg pattern:

偶數列使用Floyd-Steinberg pattern由左往右掃，奇數列則使用左右翻轉的Floyd-Steinberg pattern由右往左。

肉眼來看我自己覺得有沒有serpentine scanning差異不大，只是相比沒有serpentine scanning的Floyd-Steinberg，似乎更沒有pattern的感覺，但是某些地方卻出現了奇怪的線條（例如貓咪和後面牆壁的接縫處）。

error diffusion + serpentine scanning with Jarvis et al pattern:

同上，改為使用Jarvis et al pattern。



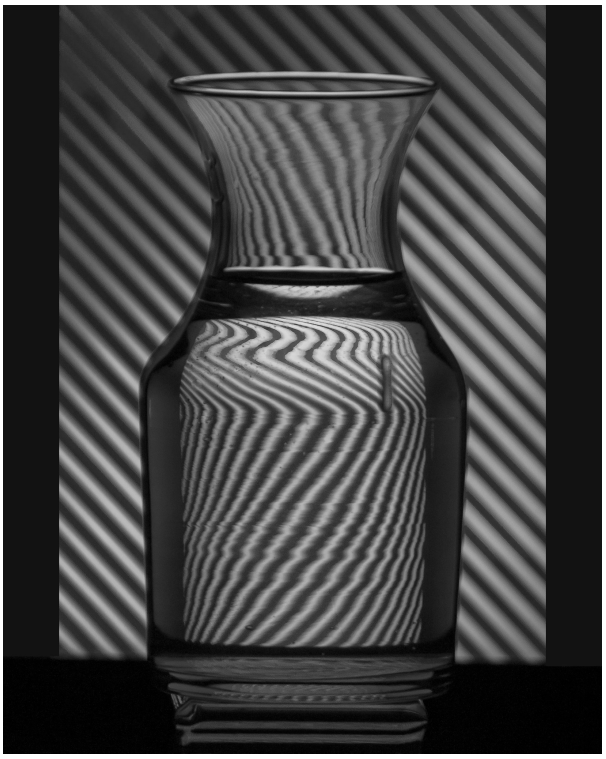
Jarvis et al pattern搭配serpentine scanning，自己覺得這次結果似乎是最好的，原圖上的許多細節都有出現，也沒有什麼pattern的感覺。

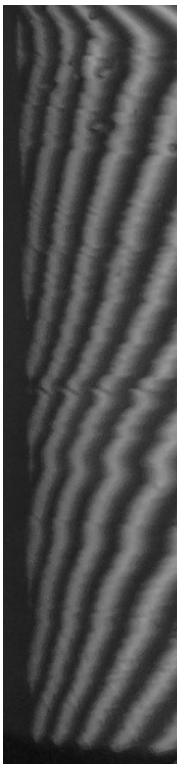
Problem 2: Frequency Domain

(a) (10 pt) By analyzing sample2.png, please **explain how to perform image sampling on it to avoid aliasing**. **Please also perform 'inappropriate' image sampling which results in aliasing in the sampled image. Output the result as result5.png**, specify the sampling rate you choose and discuss how it affects the resultant image.

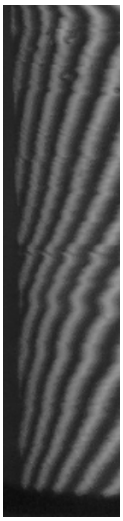
Ans: 當sampling ratio變為0.5時，sample image的結果仍然可見原本的紋路。

original image:



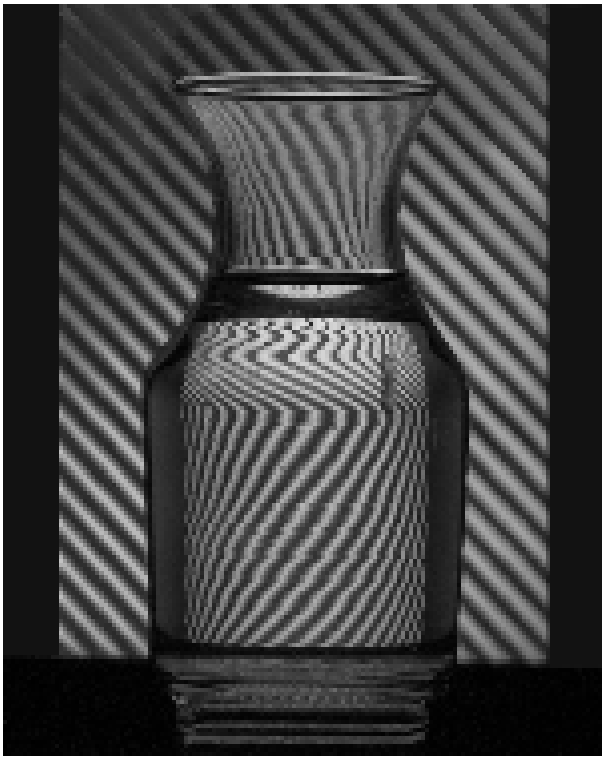


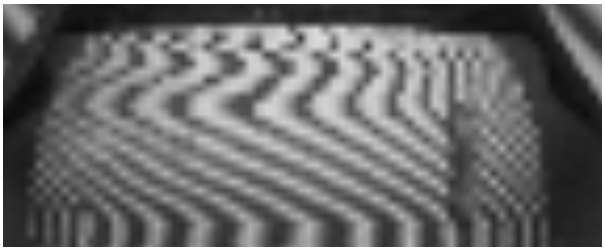
sampling ratio = 0.5



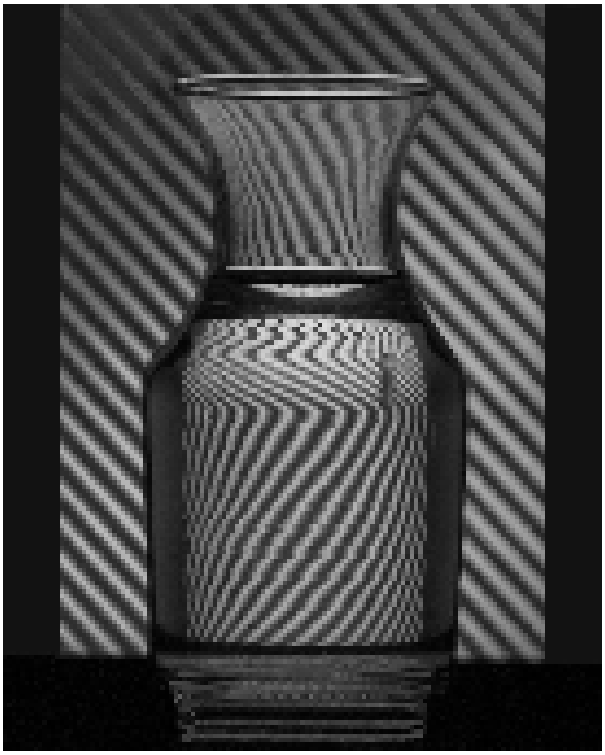
aliasing image: (sampling ratio = 0.09)

但當取點的ratio降至0.09時，比較密集的螺旋紋路變得有些與原本不相同，肉眼可見的逐漸有些接點錯誤。





aliasing image: (sampling ratio = 0.08)

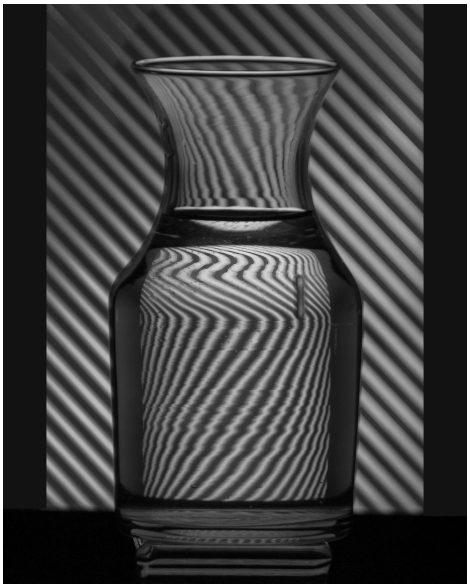


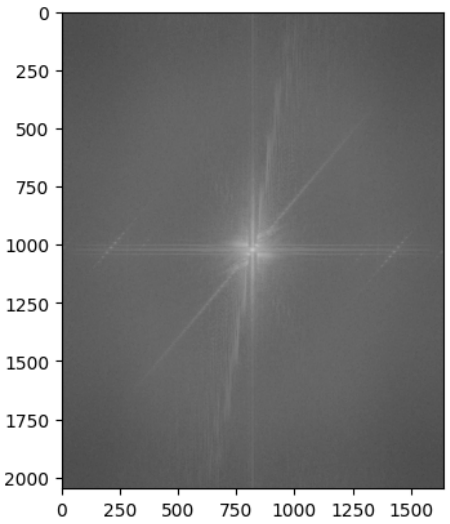
(b) (20 pt) Please perform the Gaussian high-pass filter in the frequency domain on sample2.png and transform the result back to the pixel domain by inverse Fourier transform. Save the resultant image as result6.png.

Ans:

original image:

original image in frequency domain:

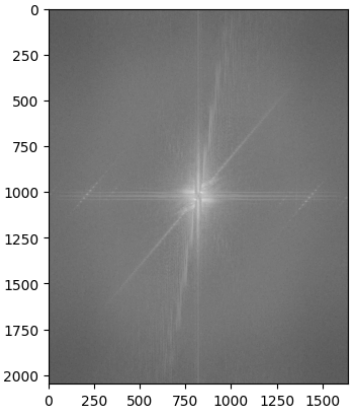


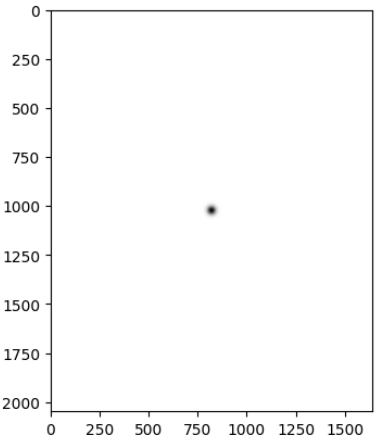


after Gaussian high-pass filters with radius = 15

after Gaussian high-pass filters with radius = 15 in frequency domain:

Gaussian high-pass filters with radius = 15

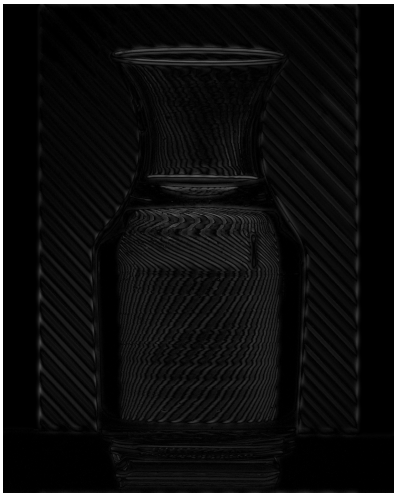




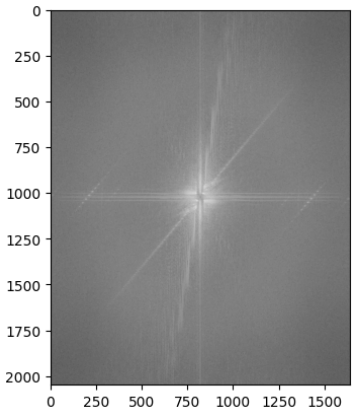
after Gaussian high-pass filters with radius = 30

after Gaussian high-pass filters with radius = 30 in frequency domain:

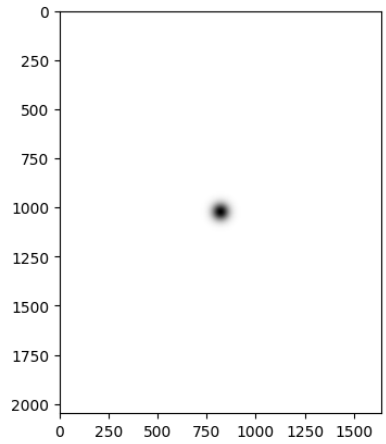
Gaussian high-pass filters with radius = 30



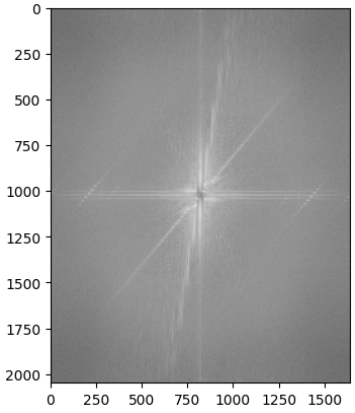
after Gaussian high-pass filters with radius = 80

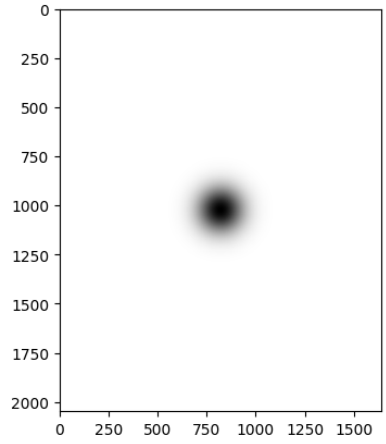


after Gaussian high-pass filters with radius = 80 in frequency domain:



Gaussian high-pass filters with radius = 80





radius愈大，mask中能通過的能通過的低頻愈少。最後選擇radius = 30，看上去輪廓線變得乾淨清晰。

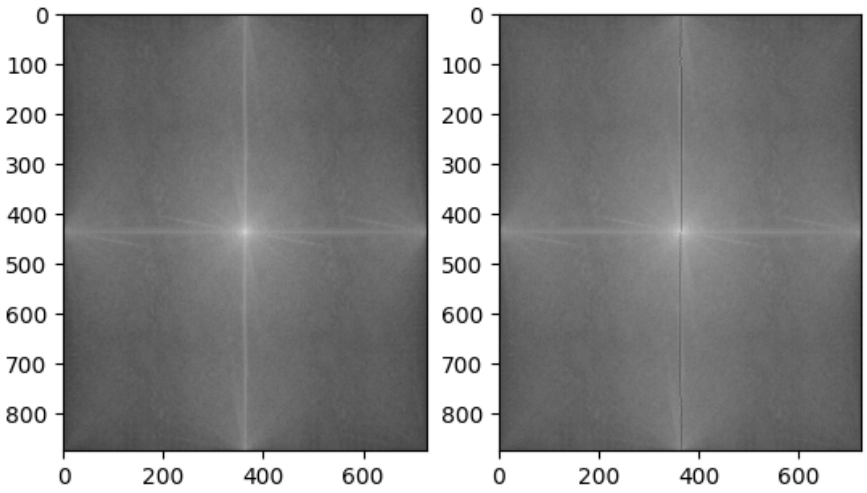
(c) (20 pt) Try to remove the undesired pattern on sample3.png with Fourier transform and output it as **result7.png**. Please also describe how you accomplished the task.

Ans:

original image:

查了一些資訊後，得知水平的水波紋應是在frequency domain中的縱軸（除了中間low frequency包含重要資訊）的部份。因此使用dp的方式找出frequency domain的vertical那條intensity最大的線，再找出horizontal 的instensity最大的部份加以保留，

在frequency domain的部份做前後比對：

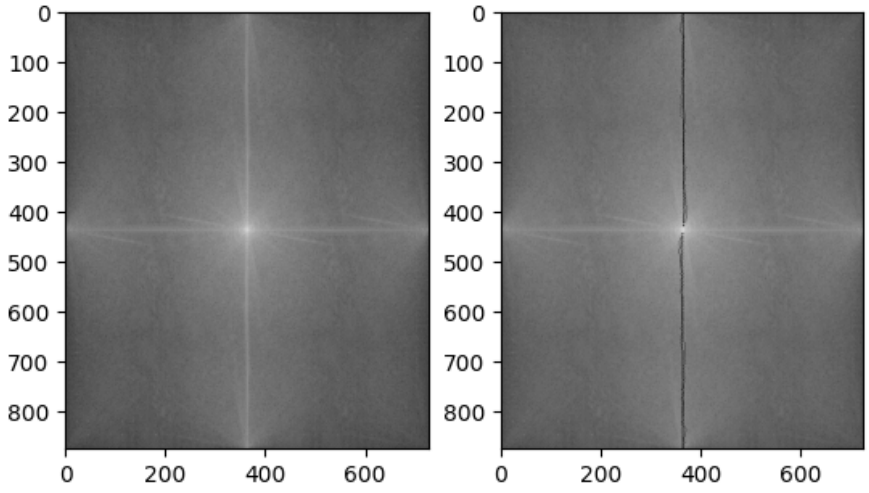


左邊為原圖的frequency domain，右邊為操作後的（垂直寬度1pixel，instensity最大的一條，保留中央重要資訊，其餘remove）

初次嘗試（只消除1pixel的寬度）

只去除frequency domain中，垂直一個pixel寬度，intensity最大的線，結果看起來不錯。
由於中心點附近的重要資訊有做保留，因此看起來沒有犧牲掉什麼細節。

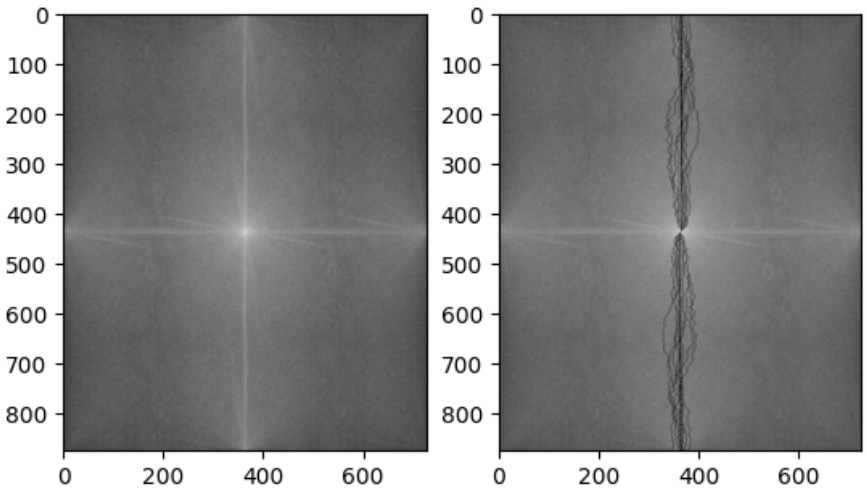
但是細看狗狗左邊的周圍還是有一些沒清乾淨的細線，因此我將垂直要移除的seam增加到3 pixel的寬度：



最好的一次

狗狗周圍的細線變少了，並且整體細節都有做保留，毛髮、亮亮的眼睛都有看得很清楚，自己覺得這是效果最好的一次。

但是當我remove的中間seam數愈多時，反而開始出現一些奇怪的光影（可能是一些細節不見了導致圖片有些空缺處，形成光影）。



效果不好的一次

附上程式碼：

```
#找出instensity最大的線條
# for "vertical seam":seam_orient='v', for "horizontal seam":seam_orient='h'
def find_largest_seam(energy_map:np.ndarray, seam_num:int, seam_orient:string):
    height,width = energy_map.shape
    max_E = np.zeros((height,width))
    max_O = np.zeros((height,width))
    seam_map = np.zeros((height,width))
    if seam_orient == 'v':
        max_E[0][:]=energy_map[0][:]
        max_O[0][:]=-1
        for i in range(1,height):
            for j in range(0,width):
```

```
                e=energy_map[i][j]
                ancestor = {}
                if j == 0:
                    ancestor[1]=max_E[i-1][j]
                    ancestor[2]=max_E[i-1][j+1]
                elif j == width-1:
                    ancestor[0]=max_E[i-1][j-1]
                    ancestor[1]=max_E[i-1][j]
                else:
                    ancestor[0]=max_E[i-1][j-1]
                    ancestor[1]=max_E[i-1][j]
                    ancestor[2]=max_E[i-1][j+1]
                max_E[i][j]=max(ancestor.values())+e
                max_O[i][j]=max(ancestor,key=ancestor.get)
    elif seam_orient == 'h':
        max_E.transpose()[0][:]=energy_map.transpose()[0][:]
        max_O.transpose()[0][:]=-1
        for j in range(1,width):
            for i in range(0,height):
                e=energy_map[i][j]
                ancestor = {}
                if i == 0:
                    ancestor[1]=max_E[i][j-1]
                    ancestor[2]=max_E[i+1][j-1]
                elif i == height-1:
                    ancestor[0]=max_E[i-1][j-1]
                    ancestor[1]=max_E[i][j-1]
                else:
                    ancestor[0]=max_E[i-1][j-1]
                    ancestor[1]=max_E[i][j-1]
                    ancestor[2]=max_E[i+1][j-1]
                max_E[i][j]=max(ancestor.values())+e
                max_O[i][j]=max(ancestor,key=ancestor.get)
    largest_k=np.array([(x,-float('inf')) for x in range(seam_num)],dtype=[('index','<i4'),('total_energy','<f4')])
    if seam_orient == 'v':
        for j in range(width):
            for k in range(seam_num):
                if max_E[height-1][j] > largest_k['total_energy'][k]:
                    largest_k[k]=(j,max_E[height-1][j])
                    break
        for k in range(seam_num):
            j=largest_k['index'][k]
            for i in reversed(range(height)):
                seam_map[i][j]=-1
                if max_O[i][j]==0:
                    j-=1
                elif max_O[i][j]==2:
                    j+=1
    elif seam_orient == 'h':
        for i in range(height):
            for k in range(seam_num):
                if max_E[i][width-1] > largest_k['total_energy'][k]:
                    largest_k[k]=(i,max_E[i][width-1])
                    break
        for k in range(seam_num):
            i=largest_k['index'][k]
            for j in reversed(range(width)):
                seam_map[i][j]=-1
                if max_O[i][j]==0:
                    i-=1
                elif max_O[i][j]==2:
                    i+=1
    return seam_map
```

```
def remove_horizontal_ripple(img):
    height,width=img.shape
    img_freq= np.fft.fft2(img)
    img_freq_center=np.fft.fftshift(img_freq)
    to_show=np.log(1+np.abs(img_freq_center))
    plt.subplot(121)
    plt.imshow(to_show,cmap='gray')
    res_freq_center=img_freq_center.copy()

    for_find_reverse_part=img_freq_center.copy()
    reserve_map=np.zeros((height,width))
    for _ in range(10):
        reserve_seam=find_largest_seam(for_find_reverse_part,1,'h')
        for_find_reverse_part[reserve_seam==-1]=0
        reserve_map[reserve_seam==-1]=-1

    for _ in range(3):
        remove_map=find_largest_seam(res_freq_center,1,"v")
        for i in range(height):
            for j in range(width):
                if remove_map[i][j]==-1 and reserve_map[i][j]!=-1:
                    res_freq_center[i][j]=0
    plt.subplot(122),plt.imshow(np.log(1+np.abs(res_freq_center)),cmap='gray')

    res_freq=np.fft.ifftshift(res_freq_center)
    res = np.fft.ifft2(res_freq)
    res = np.clip(np.abs(res), 0, 255).astype("uint8")
    return res
```