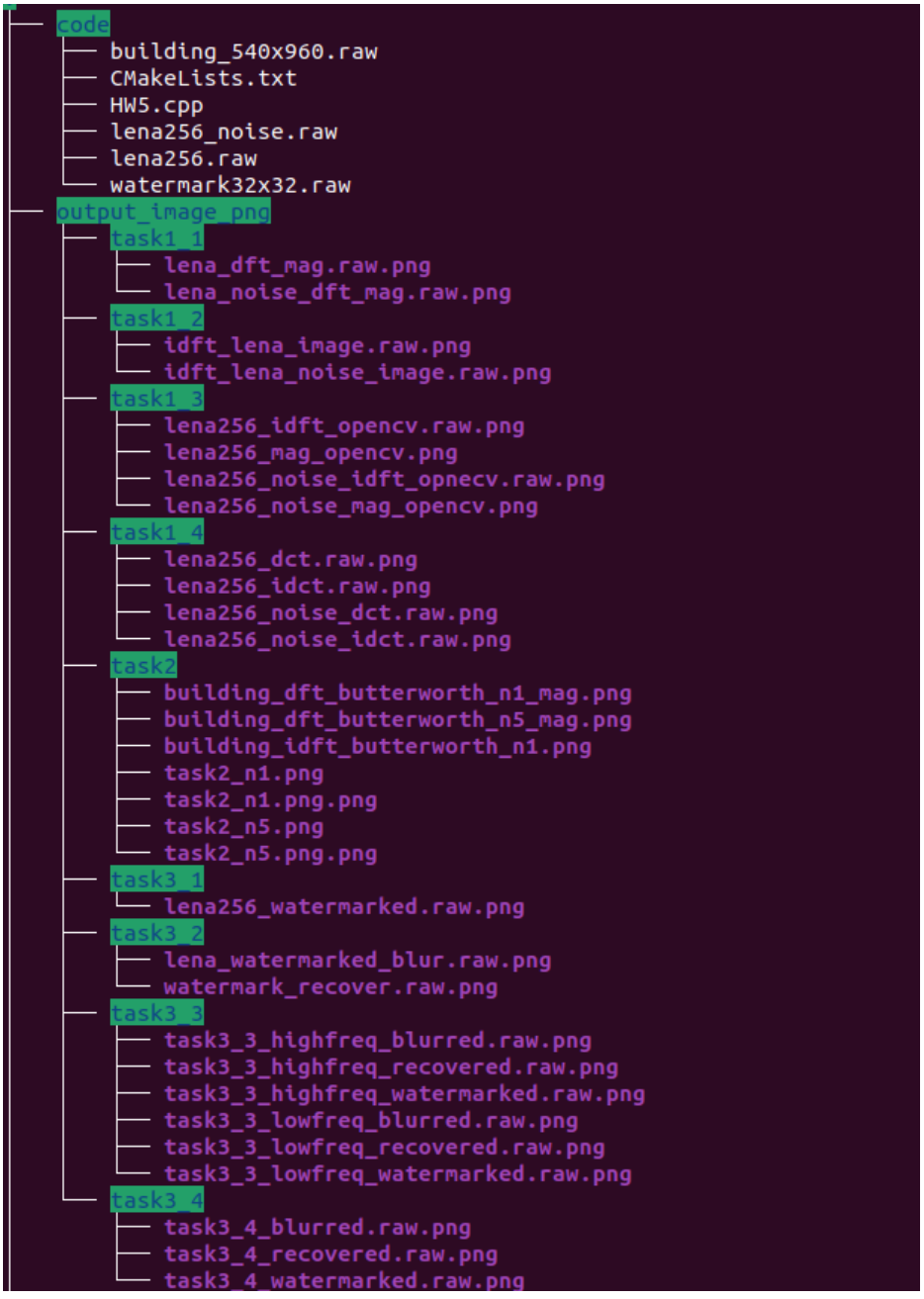
高等數位訊號處理

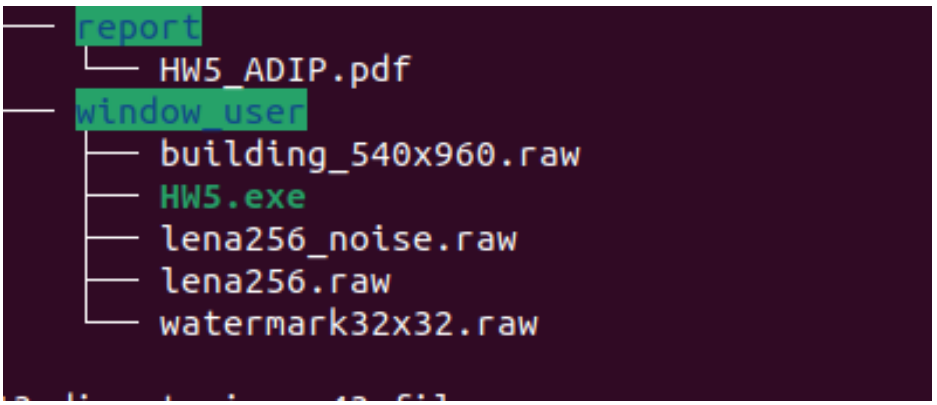
作業#(5)

姓名：闕楷宸
學號：114318047
指導老師：黃正民

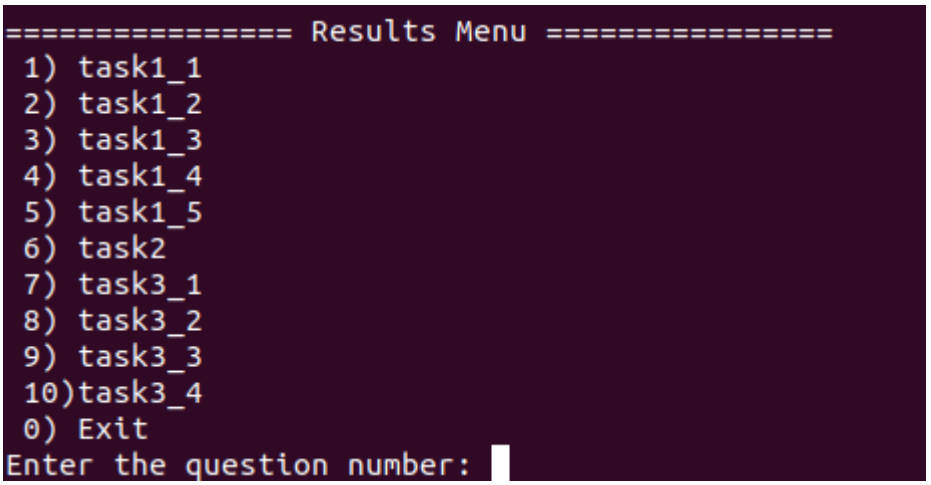
作業說明:

作業架構:


code
building_540x960.raw
CMakeLists.txt
HW5.cpp
lena256_noise.raw
lena256.raw
watermark32x32.raw
output_image_png
task1_1
lena_dft_mag.raw.png
lena_noise_dft_mag.raw.png
task1_2
idft_lena_image.raw.png
idft_lena_noise_image.raw.png
task1_3
lena256_idft_opencv.raw.png
lena256_mag_opencv.png
lena256_noise_idft_opnecv.raw.png
lena256_noise_mag_opencv.png
task1_4
lena256_dct.raw.png
lena256_idct.raw.png
lena256_noise_dct.raw.png
lena256_noise_idct.raw.png
task2
building_dft_butterworth_n1_mag.png
building_dft_butterworth_n5_mag.png
building_idft_butterworth_n1.png
task2_n1.png
task2_n1.png.png
task2_n5.png
task2_n5.png.png
task3_1
lena256_watermarked.raw.png
task3_2
lena_watermarked_blur.raw.png
watermark_recover.raw.png
task3_3
task3_3_highfreq_blurred.raw.png
task3_3_highfreq_recovered.raw.png
task3_3_highfreq_watermarked.raw.png
task3_3_lowfreq_blurred.raw.png
task3_3_lowfreq_recovered.raw.png
task3_3_lowfreq_watermarked.raw.png
task3_4
task3_4_blurred.raw.png
task3_4_recovered.raw.png
task3_4_watermarked.raw.png



主程式HW5.cpp



輸入0~10即可輸出結果

建置執行(window,linux)

Linux:

1.使用opencv 4.5.4(sudo apt install libopencv-dev)

2.建置與執行:

cd HW#5_114318047/code/

cmake -S . -B build

cmake --build build -j

./build/HW5

Window:

cd window_user

./HW5.exe

Window 建置:

1:Cmake:

winget install Kitware.CMake

2:OpenCV for window:

C:\opencv\build\x64\vc16\bin(make sure DLLs are in this dir)

3:Confirm CMake config file exists:

C:\opencv\build\x64\vc16\lib\OpenCVConfig.cmake

4:Build and run

cd HW#5_114318047\code

5:Configure:

cmake -S . -B build -G "Visual Studio 17 2022" -A x64 -DOpenCV_DIR="C:\opencv\build\x64\vc16\lib"

6:Produce exe:

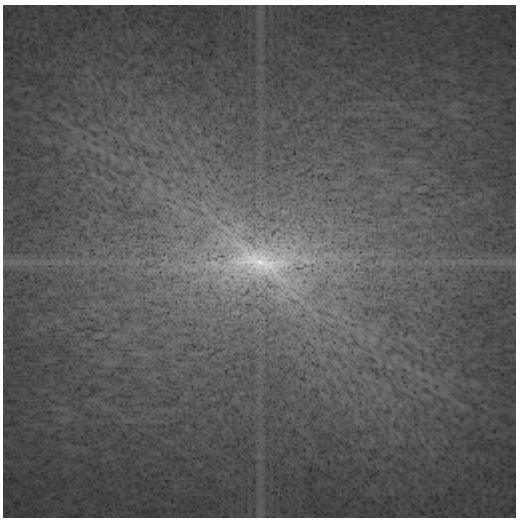
cmake --build .\build --config Release

7:Run exe

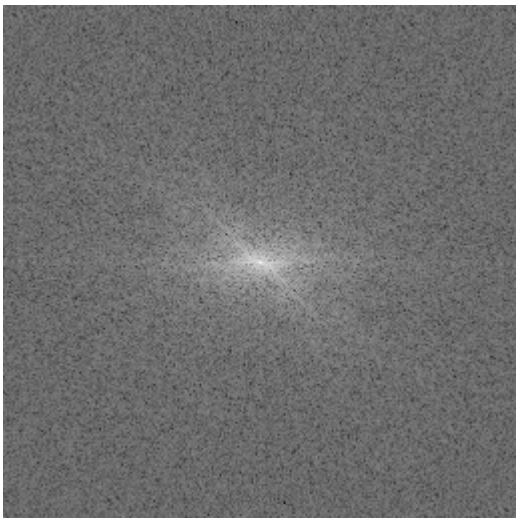
.\build\Release\HW5.exe

1-1:DFT

Write your own DFT subroutine (with origin shifted) and test on lena256.raw and lena256_noise.raw. Show the magnitude spectrum output with contrast enhancement to see the spectrum details.
Discuss the spectrum differences between the two results



Magnitude of lena256.raw
(lena_dft_mag.raw.png)



Magnitude of lena256_nosie.raw
(lena_noise_dft_mag.raw.png)

Discussion(1-1)

Lena256.raw　　lena256_noise.raw

```
Enter the question number: 1
task1_1 done: magnitude saved.
Custom DFT time: 2776 ms
```

從以上Spectrum 跟原圖可以發現原圖(lena256.raw)在頻域上在中心強度更強，並且十字線更為明顯，這是因為原圖(lena256.raw)人物上有許多細節，比如說臉的輪廓，或是頭髮、身體等等，而亮度都集中在中心是因為影像的大部分能量都在低頻，而最後在頻譜上的斜線則是對角方向的人物特徵變化。

而加入雜訊的圖像(lena256_noise.raw)在頻譜中的表現則是很明顯亮度是很均勻的，因為噪點就是高頻，而頻譜中的十字線也較不明顯。

在程式上則是對row、column進行1-D DFT，並且在2-D DFT 中的流程則是將原點從左上移至圖像中心(W/2,H/2)，並且對row進行DFT，再對column進行DFT，其中需要注意的點則是行運算時，需要將對row進行DFT的結果進行抽取，並asssign給col vector進行DFT，因為各column中並不連續，不像列運算一樣：

```
        for (int x = 0; x < M; x++)
            col_in[x] = rowDFT[x][y];

        // 1D DFT on column
        DFT_1D(col_in, col_out, M);
```

而實際運算則是按照講義中的DFT公式進行撰寫：

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)e^{-j2\pi\frac{ux}{M}} \qquad u = 0, 1, 2, \ldots, M-1,$$

以下以1-D DFT舉例：

需要注意的點則是需要使用尤拉公式轉換成cos與sin的組合進行，並且學習要如何使用std::complex，與2-D complex vector的操作方式

```
    for (int u = 0; u < M; u++) {
        std::complex<float> sum(0,0);

        for (int x = 0; x < M; x++) {
            float angle = -2.0f * M_PI * u * x / M;
            sum += f_x[x] * std::complex<float>(cos(angle),
sin(angle));
        }
        f_u[u] = sum / (float)M;
    }
```

1-2:DFT

Write your own IDFT subroutine and test it on DFT outputs from Problem 1.1.Show the result images.Check whether your output images can be exactly transformed back to the original image or not.

idft_lena_image.raw.png

idft_lena_noise_image.raw.png

```
Enter the question number: 2
task1_2 done: IDFT restored images saved.
Custom IDFT time: 2742 ms
PSNR between original and IDFT-restored image = 51.1423 dB
PSNR between original noisy and IDFT-restored noisy image = 51.1529 dB
```

Discussion (1-2)

此題與task1-1相似，只須先準備好1-D IDFT與 2-D IDFT即可，而在task1-1時，我使用call by reference 去存取DFT結果，因此此題只須指向該位址進行存取即可，以下為task1_2的片段，以及task1_1的function call，dft_lena為lena256.raw做完的結果，dft_lena_noise則為lena256_noise.raw的結果

```
// Do IDFT based on DFT results from task1_1
IDFT_2D(dft_lena, img, H, W);
IDFT_2D(dft_lena_noise, img_noise, H, W);
```

```
task1_1("lena256.raw", "lena256_noise.raw", dft_lena,
dft_lena_noise);
```

以下以1-D IDFT舉例：

注意事項與1-D DFT相似

$$f(x) = \sum_{u=0}^{M-1} F(u) e^{j2\pi \frac{ux}{M}} \qquad x = 0, 1, 2, \ldots, M-1.$$

```
for (int x = 0; x < M; x++) {
    std::complex<float> sum(0,0);

    for (int u = 0; u < M; u++) {
        float angle = 2.0f * M_PI * u * x / M;
        sum += f_u[u] * std::complex<float>(cos(angle),
sin(angle));
    }
    f_x[x] = sum;
}
```

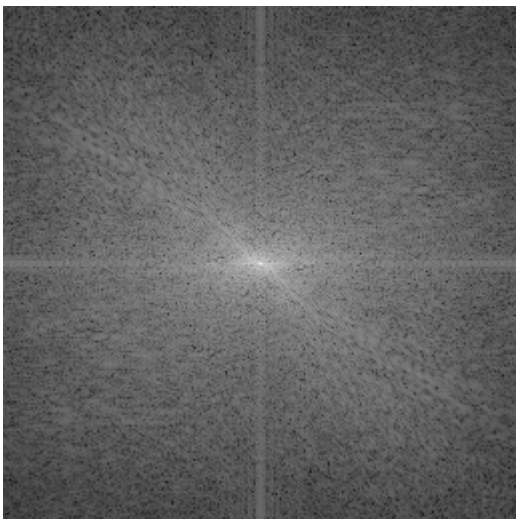
此外，此題引入PSNR計算的依據，以此比較原始圖像(ex:lena256.raw)與restored image(idft_lena_image.raw.png)的差異，因為肉眼看不出差異，而最後的結果可以發現最後的PSNR也不是無限大，仍有損失(PSNR=51db)

1-3 DFT
Use the built-in FFT and IFFT functions in OpenCV to process these two images. Discuss the execution time differences between Problems 1.1 and 1.2.Discuss the difference in the result and speed.The DC component should be centered.

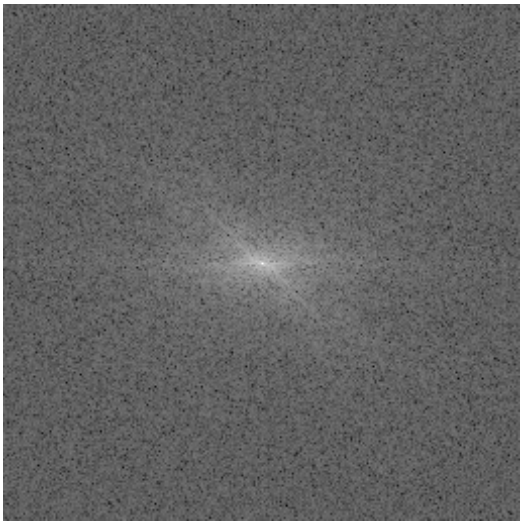
```
Enter the question number: 3
task1_3: OpenCV FFT + IFFT done.
OpenCV DFT time: 8 ms
OpenCV IDFT time: 3 ms
PSNR between original and OpenCV IDFT-restored image = inf dB
PSNR between original noisy and OpenCV IDFT-restored noisy image = inf dB
```

```
Enter the question number: 1
task1_1 done: magnitude saved
Custom DFT time: 2713 ms
```

```
Enter the question number: 2
task1_2 done: IDFT restored images saved
Custom IDFT time: 2775 ms
```



lena256_mag_opencv.png
(lena256.raw的結果)



lena256_noise_mag_opencv.png
(lena256_noise.raw的結果)

lena256_idft_opencv.raw.png

lena256_noise_idft_opnecv.raw.png
(IDFT的結果)

Discussion (1-3)

先比較執行時間，task1_1為2713ms，task1_2為2775ms，而OpenCV的DFT只須8ms，IDFT也只須2ms，而且task1_1與task1_2的PSNR為51db，但是OpenCV 的PSNR則為理想值(無限大)，雖然從輸出圖像無法看出差別，但是從執行時間以及重構上有很大的差異。

以下是我認為的原因:
第一(execution time):
我使用的1-D DFT的時間複雜度為O(N^2)，並且在2-D DFT執行了兩次1-D DFT
rowDFT:O(MxN^2)
colDFT :O(NxM^2)

Total: O(M^3+N^3)
在256x256圖像中，則須(O(256^3+256^3) = O(33,554,432)
並且IDFT再做一次，因此複數運算次數達到6700萬多次

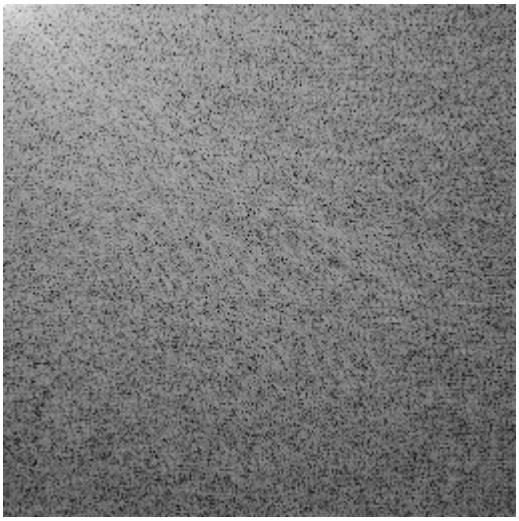
OpenCV所使用的FFT，並且叫做Coolkey-Tokey FFT，時間複雜度則只有O(NlogN)，在256z256中只須O(N^2logN)=>O(524,288)

第二(PSNR):
我自己的DFT以及IDFT使用std::clamp()會造成誤差，並且我IDFT分階段處理，在使用sin、cos計算浮點時會造成誤差。

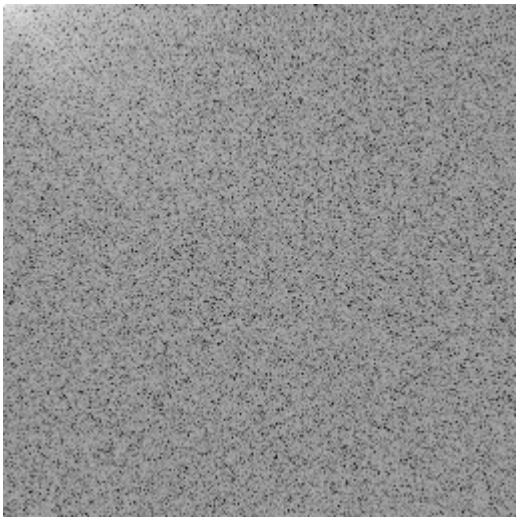
1-4 DFT

Use the built-in DCT and IDCT functions in OpenCV to process these two images(clean and noisy version) by performing a full-image DCT(not the traditional

8x8 block DCT).Show the DCT spectra with contrast enhancement.



lena256_dct.raw.png



lena256_noise_dct.raw.png

lena256_idct.raw.png

lena256_noise_idct.raw.png

Discussion (1-4)

此題使用OpenCV提供的DCT以及IDCT進行運算，我們從結果的spectrum 可以看出經過DCT處理的結果，有雜訊的lena在亮度上更均勻，以下為DCT操作細節:

第零:使用OpenCV DCT

```
cv::dct(img_float, dct_img);
cv::dct(img_noise_float, dct_img_noise);
```

第一: 使用abs找尋magnitude

```
cv::Mat mag = cv::abs(dct_img);
cv::Mat mag_noise = cv::abs(dct_img_noise);
```

第二:使用log(1+DCT)進行壓縮

```
cv::log(mag, mag);
cv::log(mag_noise, mag_noise);
```

第三:使用gamma correction強化暗細節

```
double gamma = 0.4;
cv::pow(mag, gamma, mag);
cv::pow(mag_noise, gamma, mag_noise);
```

第四:normalization

```
cv::normalize(mag, mag, 0, 255, cv::NORM_MINMAX);
cv::normalize(mag_noise, mag_noise, 0, 255, cv::NORM_MINMAX);
```

此外，我們可以從DCT公式中發現他與先前DFT的差異:
第一:DFT使用尤拉作為基底，DCT使用cos，因此沒有複數的問題
第二:DCT沒有負的頻率，因此不需要進行shifting

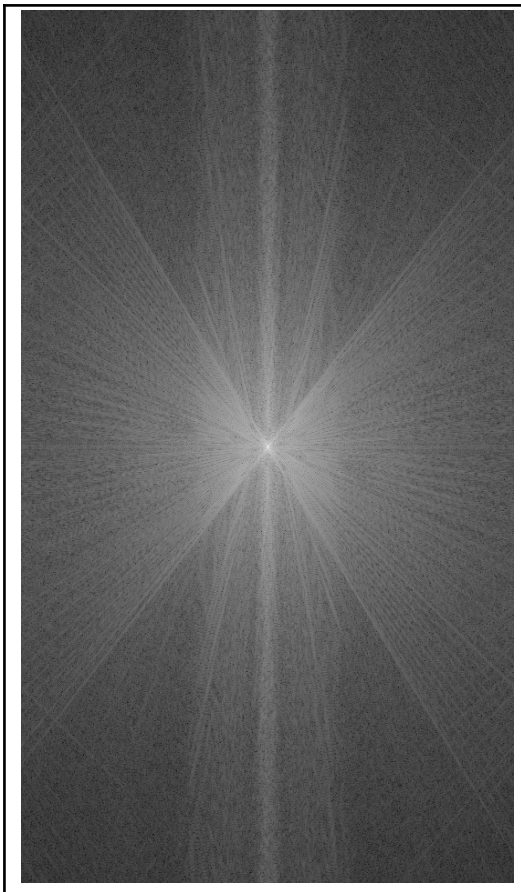
$$F(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{\pi(2x+1)u}{2N}\right], \quad u = 0, \ldots, N-1$$

:

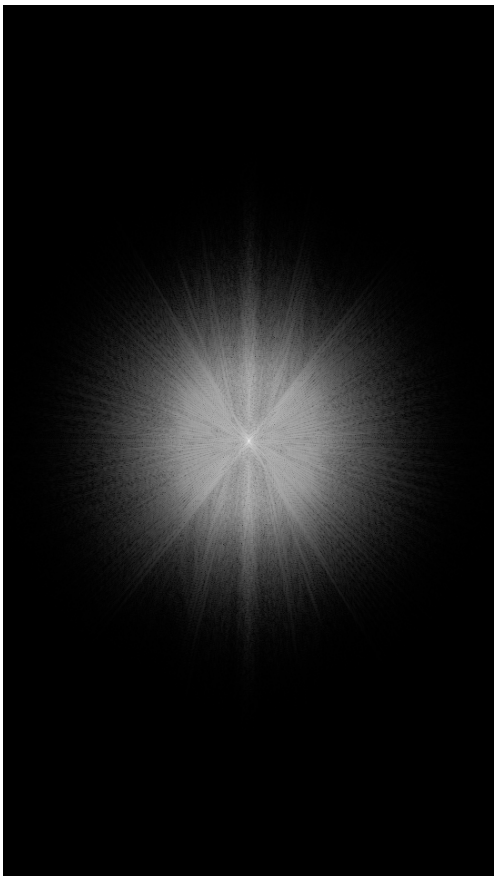
$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}}, & u = 0 \\ \sqrt{\frac{2}{N}}, & u > 0 \end{cases}$$

1-5
Discuss the differences between DFT and DCT.Explain why JPEG compression uses DCT instead of DFT?
Discussion (1-5)
使用DCT進行JPEG壓縮有以下幾個原因: 1. DCT 只需要處理實數，因此可以減少許多儲存空間，與降低複雜度 2. DCT大部分的能量都集中在左上角，但是DFT的能量分散，導致不容易進行壓縮，品質變差 3. DFT在處理時會假設圖像邊界首尾相連，形成週期性訊號，但是實際上邊緣差異大，因此會在邊界上發生高頻訊號(高頻不是在圖像中發生，而是邊緣) 4. DCT在處理圖像中使用鏡像的方式將邊界延伸，不會造成DFT的高頻訊號發生

2:Filtering in frequency domain
Apply Butterworth Lowpass Filter on building_540x960.raw in the frequency domain.Use D0=80 and n=1 and 5 for two lowpass filters.For both cases(n=1&5),display the magnitude spectrum and the output image produced by the IDFT(You can use OpenCV's or your own DFT and IDFT functions for this problem).Compare and discuss the visual difference among the resulting images,focusing on how varying order frequencies influence the clarity,detail,smoothness,ripple,...etc. of the filtered images



building_dft_butterworth_n1_mag.png



building_dft_butterworth_n5_mag.png

Building_idft_butterworth_n1.png

building_idft_butterworth_n5.png

Discussion (2)

以下分成Magnitude spectrum與經IDFT之圖像進行比較:

Magnitude:

n=1 butterworth:

濾波器對高頻抑制較弱，因此從頻譜上還是看得到許多線條(高頻)

n=5 butterworth:

濾波器對高頻抑制較強，因此從頻譜上來看接近理想低通的特性。

Spatial domain:

n=1 butterworth:

由於低頻跟高頻都被保留，因此建築物的邊緣仍可看見，整體不會過度模糊

n=5 butterworth:

高頻被濾除，因此邊緣模糊，整體看起來霧霧的。

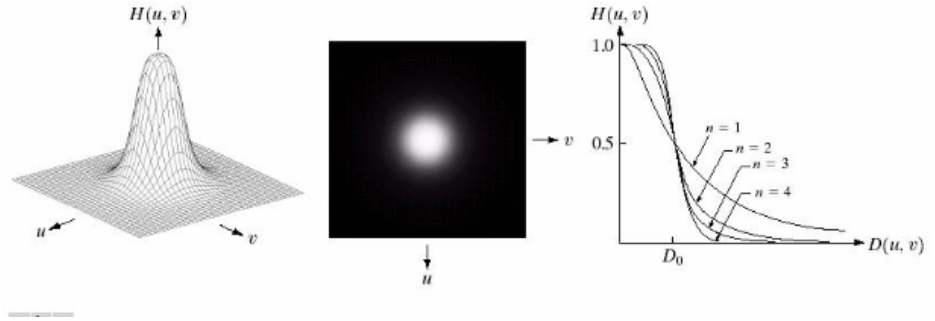
building_540x960.raw

building_idft_butterworth_n1.png

building_540x960.raw

building_idft_butterworth_n5.png

$$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}}, \quad \text{where } n \text{ is filter order}$$



我們也可以從課本的波形看出，當n增加時，曲線越陡，過渡區域變窄，低頻與高頻邊界明顯。

3-1:Watermark simulation

First, the image lena256.raw is read and divided into 8x8 blocks denoted as Ci, resulting in 1024 total blocks.In parallel.serializing the watermark32x32.raw into a 1D bitstream wi.Each bit of this watermark will be embedded into one corresponding 8x8 block of the image.

lena256_watermarked.raw.png

```
Enter the question number: 7
PSNR between original Lena and watermarked Lena = 39.3546 dB
task3_1 done: Watermark embedded into Lena.
```

Discussion(3-1)

此題按照題目步驟依序執行，分別是將pixel值轉成float format，然後進行DCT，再來是提取出A=C(2,3),B=C(3,2)，並使用下述方法進行:

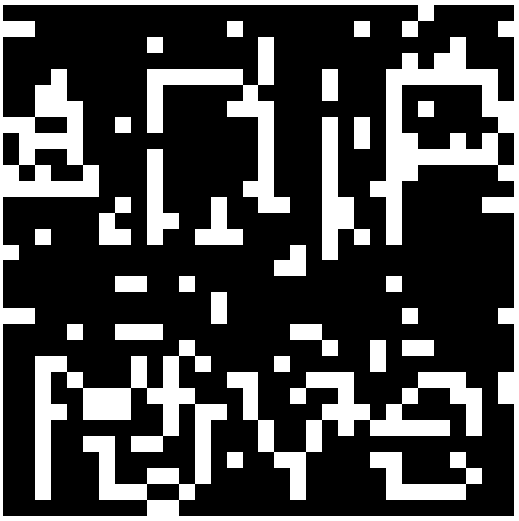
```
* If the watermark bit wi =1 and A < B,
    then enforce A > B by setting:
        C(2,3) = B + 10
* If the watermark bit wi =0 and A >= B,
    then enforce A < B by setting:
        C(3,2) = A + 10
```

最後做完IDCT完之後輸出lena256_watermarked.raw.png，從肉眼上看不出與原圖的區別，但在進行PSNR運算後還是可以發現與原圖之間的差異(39.5db)

3-2: Watermark simulation

Next,apply a 3x3 Gaussian blur to the lena_watermarked.raw image to simulate the distortion effects of compression,saving the output as lena_watermarked_bur.raw.Following this,use your implemented extraction method to restore the watermark from this new blurred image.Save and show the result as watermark_recover.raw.After recovering the watermark,did you notice if certain areas are particularly susceptible to interference,causing the letters to be corrupted? Explain the reason?

lena_watermarked_blur.raw.png



watermark_recover.raw.png

```
Enter the question number: 8
PSNR between original watermark and recovered watermark = 9.1339 dB
task3_2 done: blurred image + recovered watermark saved.
```

Discussion(3-2)

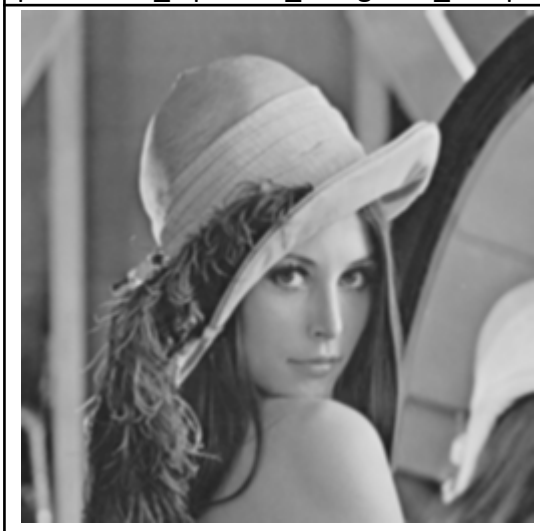
此題將在task3-1的輸出圖像(lena_watermarked.raw)進行3x3 gaussian filter，做法與先前作業4類似，做完處理後儲存成lena_watermarked_blur.raw，並且使用做完處理的圖像進行recover，實際作法為第一:將被模糊的圖像轉成float，並且對他進行DCT，並且在C(2,3)以及C(3,2)的位置進行讀取A以及B，最後再用當初embed的方法進行extract，(if A > B bit = 1; else bit = 0)，即可。

至於從輸出圖像可以發現進行3x3 Gaussian會把高頻濾掉，但是watermark是藏在中頻的位置，所以不會被完整濾除，但是像文字的邊緣(高頻)就會被破壞，而且因為做完3x3 Gaussian後A跟B的數值都會變小，差距也變小，因此在原本A > B的地方就有可能變成 A < B，導致出現破碎的情況。

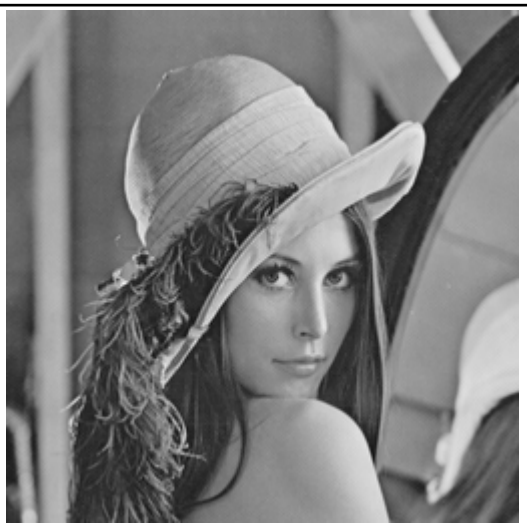
```
            int wi = watermark_bits[bit_index++];

            if (wi == 1 && A < B)
                block_dct.at<float>(2, 3) = B + 10;

            else if (wi == 0 && A >= B)
                block_dct.at<float>(3, 2) = A + 10;
```

3-3:Watermark simulation

Use each laplacian result to sharpen the image and produce parthenon_laplacian_4neighbor_sharpened.raw and parthenon_laplacian_8neighbor_sharpened.raw



task3_3_highfreq_blurred.raw.png

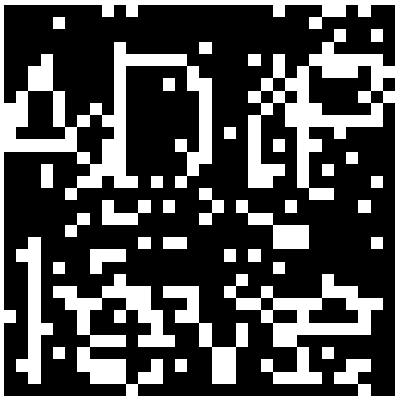


task3_3_highfreq_watermarked.raw.png

task3_3_lowfreq_blurred.raw.png

task3_3_lowfreq_watermarked.raw.png

task3_3_highfreq_recovered.raw.png



task3_3_lowfreq_recovered.raw.png

```
Enter the question number: 9
PSNR of recovered watermark (low-frequency embedding) = 8.42983 dB
PSNR of recovered watermark (high-frequency embedding) = 5.93659 dB
task3_3 done: compare low-frequency vs high-frequency watermark embedding.
```

Discussion(3-3)

此題按照提說所示，需要將低頻C(1,2),C(2,1)以及高頻C(6,7),C(7,6)embed進圖像中並進行模糊處理，並且再透過相同的方式extract出來，我們可以很明顯的從結果圖來看recover的結果，在低頻C(1,2),C(2,1)embed經過gaussian再extract出來的效果好很多，雖然也有產生破碎的圖像，但是輪廓還是看得出來，證實在低頻塞入watermark再經過low pass filter對watermark影響不像塞在高頻影響來得大，在最後PSNR也可以看出塞入低頻的結果recover完之後PSNR也比塞入高頻再recover較高。

3-4:Watermark simulation

Present the edge detection results and the sharpened images,and discuss the difference between the two filter configurations in terms of characteristics and sharpening effects.

task3_4_recovered.raw.png

```
Enter the question number: 10
Task3_4: PSNR = 38.0904 dB
task3_4 done: improved watermarking with margin.
```

Discussion(3-4)

原先，我有嘗試過將task3-1所使用的方式，只是增加DELTA，效果就有明顯改善，如下圖，但是當我越調越大後，後來的問題就變成他會導致圖像失真，與原本watermark相差越來越遠

將+10改成+40　　+10

```
int wi = watermark_bits[bit_index++];

if (wi == 1 && A < B)
    block_dct.at<float>(2, 3) = B + 10;

else if (wi == 0 && A >= B)
    block_dct.at<float>(3, 2) = A + 10;
```

因此，我嘗試了其他種方式，也就是後來所使用的方法，使用一個區間 `|A - B| >= DELTA`，這個方式可以使A,B之間保持最小的差異，還有一個關鍵點是我使用A跟B的平均值`m=0.5(A+B)`在`embed`方法上扮演很重要的工作，`embed`邏輯：

```
If A > B, A' = m+DELTA/2, B' = m-DELTA/2
   A < B, A' = m-DELTA/2, B' = m+DELTA/2
```

平均值m可以讓embed的時候確保A+B(修改之前)，A'+B'(修改之後)
=(m+DELTA/2)+(m-DELTA/2)=2m=A+B，保持不變(A'+B'=A+B)
所以就可以確保A跟B在圖像的亮度跟對比上不會有太大的影響。

本次作業繁瑣，助教辛苦了