ECSE 316 Assignment 2 report

April 18th, 2020

Programming 36
260781081 Liang Zhao
260712639 Yudi Xie

1. Design

1.1. The Discrete Fourier Transform and its inverse method

These two methods are straightforward to implement if we are given the formula Figure 1.1-1:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{i2\pi}{N} kn}$$

Figure 1.1-1

And its inverse shown in Figure 1.1-2:

$$x_n = \frac{1}{N} \sum_{n=0}^{N-1} X_k e^{\frac{i2\pi}{N} kn}$$

Figure 1.1-2

1.2. Converting the image to its Fast Fourier Transform (FFT) by Cooley-Tukey

We implement FFT by using divide and conquer design paradigm. We split the Discrete Fourier Transform into Odd and Even parts recursively until only 1 element in each sub-FT. We solve each individual small FT and sum them back together. The formula follows Figure 1.2-1

$$X_{even} + e^{\frac{-i2\pi}{N} k} X_{odd}$$

Figure 1.2-1

1.3. Denoising the image

We take the FFT of the image and set all the high frequencies to zero before inverting to get back the filtered original. After many trials we found that filtering out range between 0.932pi and 1.072pi will make the image look the best.

1.4. Compressing the image

We compress the image by setting some Fourier coefficients to zero. We will always set Fourier coefficients with high frequency to zero first. We implement 6 different compression levels 0%, 20%, 40%, 60%, 80%, 95%. For example, when the compression level is 20%, we will set range 0.8pi to 1.2pi to 0.

1.5. The Two-Dimensional Fourier Transform

We used nested function to implement 2D FFT with the following formula in Figure1.5-1

$$\begin{aligned} F[k,l] &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[m,n] e^{\frac{-i2\pi}{M} km} e^{\frac{-i2\pi}{N} ln} \\ &= \sum_{n=0}^{N-1} \Big(\sum_{m=0}^{M-1} f[m,n] e^{\frac{-i2\pi}{M} km} \Big) e^{\frac{-i2\pi}{N} ln} \\ &\text{for } k = 0, 1, \ldots, M-1 \text{ and } l = 0, 1, \ldots, N-1. \end{aligned}$$

Figure 1.5-1

We calculate the inner summation first, which is 1D Fourier Transform. We can calculate it by using 1D FFT. Next, we calculate the outer summation, which can be done by using a nested function. The FFT method outputs a 2D array, which becomes the inputs of another FFT method.

2. Testing

2.1. The Discrete Fourier Transform and its inverse method

We can check if the image in converted to its DFT successfully be inversing it back to the original image.

2.2. Converting the image to its FFT

We implemented an inverse Fourier Transform method to check if the FFT is converted successfully. If it is successful to convert, then we would be able to convert the FFT back to the original image. We could also compare the runtime of FFT to naïve approach to check if our FFT works.

2.3. Denoising the image

We display the denoised graph so that we could pretty straightforward to checking the effect of denoising.

2.4. Compressing the image

This method is pretty straightforward in implementing. We could have a direct feedback by displaying the compressed images.

2.5. The Two-Dimensional Fourier Transform

This is one of the most important algorithms. We depend on a functional 2D FF to have a direct view of our implementation of FFT, Denoising and Compressing. Luckily, we have already known the formula so we could implement it in a straightforward way. And because discreate Fourier Transform is easy to implement, so we could inverse the image with DFT forth and back to check if it dis**played** as expected.

3. Analysis

For naïve DFT, the runtime of the naïve DFT for 1D is $O(N_2)$, N is the length of the array. This is because for every element in the 1D array we must do the summation and Each summation has N components.

For 2D DFT, we replace N to N_2. Because it is now an array with N_2 elements. Therefore, the time complexity of the 2D DFT is $O[N_4]$.

For FFT, because we used merge sort to implement this method, so its runtime complexity should be close to merge sort which is O(NlogN), where N is the number of elements in the array.

For 2D FFT, it is essentially an 1D array with N_2 elements. Because the complexity of 1D FFT is O[N*logN], we replace N to N_2, as the number of elements in 2D array is N_2. Thus, the complexity is $O[N_2*\log(N_2)]$. We can simplify this to $O[N_{2*}2\log(N)]$ which shows the time complexity of the 2D FFT is $O[N_2\log N]$.

4. Experiment

4.1. FFT

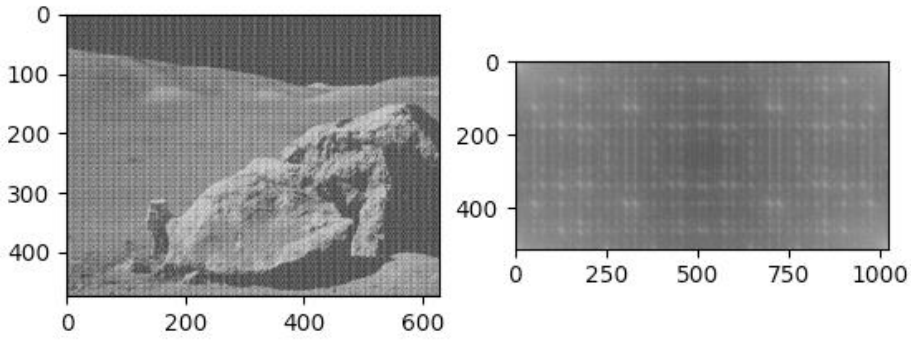


Figure 4.1-1 2D FFT by our implementation

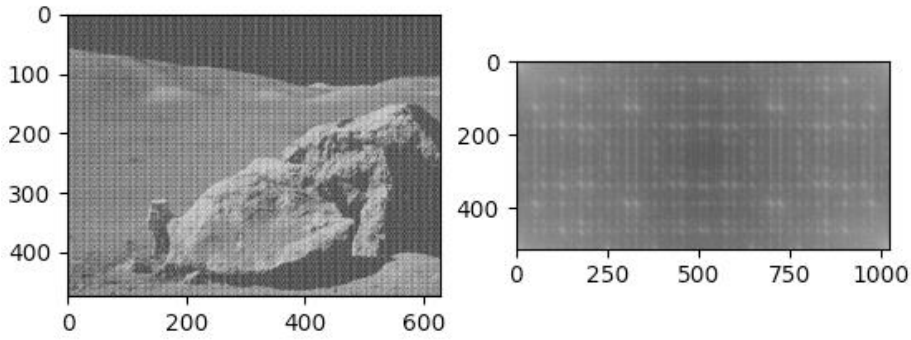


Figure 4.1-2 2D FFT by np.fft.fft2

4.2. Denoising

4.2.1 Denoising high frequencies.

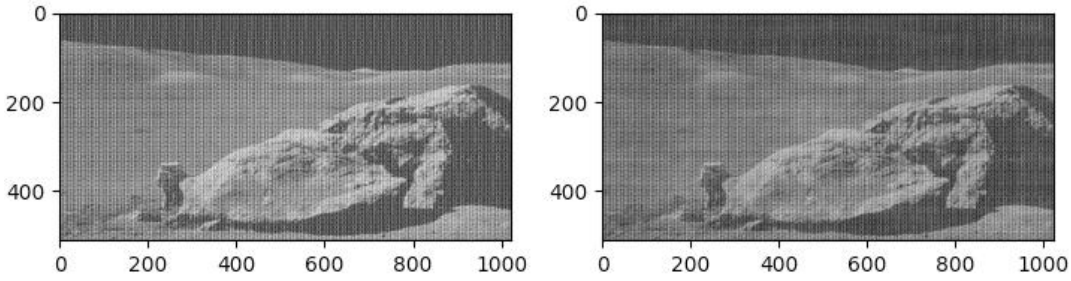


Figure 4.2-1 Denoising by filtering out high frequency range 0.932pi to 1.072pi, this looks the best

4.2.2 Denoising low frequencies.

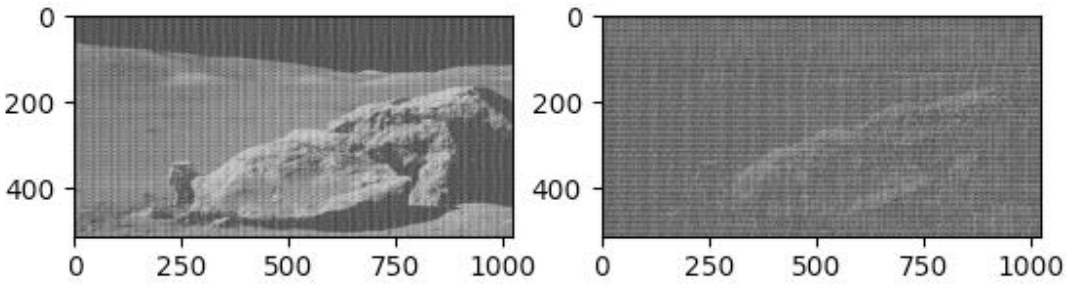
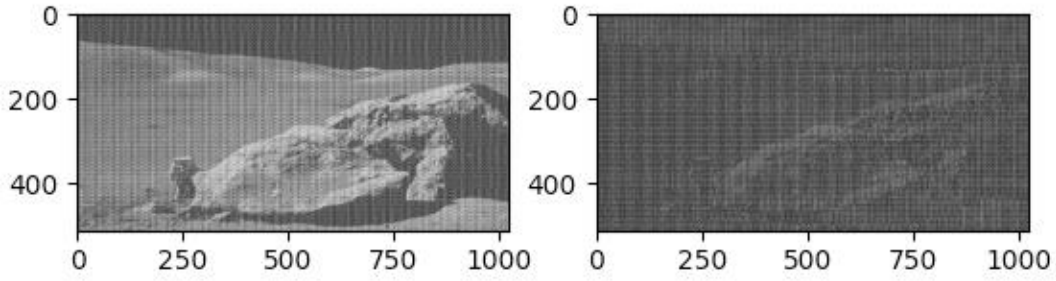


Figure 4.2-2 Denoising by filtering out low frequency between 0.45pi and 0.55pi

4.2.3 Denoising 10% of low frequencies and 10% of high frequencies



From these images we notice that by denoising high frequencies it will not change the image much but by denoising low frequencies only some details are blurred.

4.3 Compression

Starting from the highest frequencies, we gradually increase the range of compressing to 95%

1. The original image has 524,288 non-zero elements
2. Set range 0.8pi to 1.2pi to zero: # Non-zero elements: 418809
3. Set range 0.6pi to 1.4pi to zero: # Non-zero elements: 313610
4. Set range 0.4pi to 1.6pi to zero: # Non-zero elements: 208702
5. Set range 0.4pi to 1.6pi to zero: # Non-zero elements: 104289
6. Set range 0.05pi to 1.95pi to zero: # Non-zero elements: 26256

The corresponding images are shown in Figure 4.3-1

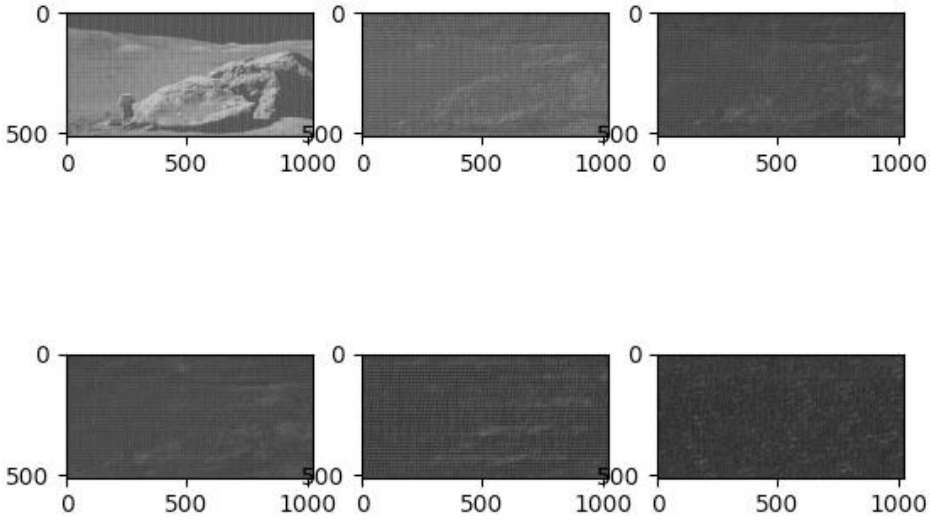


Figure 4.3-1

4.4 Plotting for runtime

The following graph Figure 4.4-1 shows that we have 4 tests with 32, 64, 128 and 256 pixel height images. In the graph we see that FFT is much faster than naive Fourier Transform and inverse FFT is slight faster than FFT. This difference grows larger as the number of elements grows.

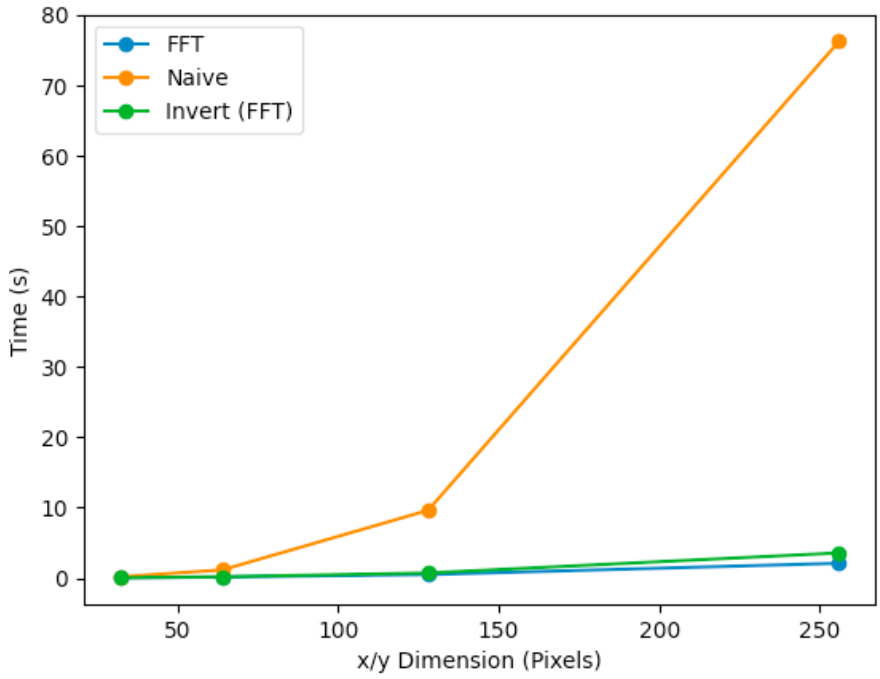


Figure 4.4-1