Lab 5

Use two images for each operation to do the following operations and write down their advantages and disadvantages and explain your results:

1. Implement 2D DFT for all provided images and analyze their transformed images in the frequency domain
2. Reconstruct lena.pgm only using the phase angle from 2D DFT
3. Reconstruct lena.pgm only using the magnitude from 2D DFT

Algorithm:

1. Discrete Fourier Transform

$$F(u,v) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)\, e^{\frac{-j2\pi um}{M}} e^{\frac{-j2\pi uv}{N}}$$

Where F the image after taking Fourier Transform, f is the original image

Input: Input: an unsigned char array that stores the source image data

Output: an unsigned char array that stores the output image data

Shift image f, $f(m,n) = (-1)^{m+n} f(m,n)$

Create two zero image t and g, where Rt is the real part of t, It is the imaginary part of t, and the same as g.

```
for (i = 0; i < image. Height; i++) {
    for (j = 0; j < image. Width; j++) {
        for (m = 0; m < image. Height; m++) {
            theta = (-2) * PI * (i * m) / image.Height;
            Rt[i][j] += (int)tempin[m][j] * cos(theta);
            It[i][j] += (int)tempin[m][ j] * sin(theta);
        }
    }
}

for (i = 0; i < image->Height; i++) {
    for (j = 0; j < image->Width; j++) {
        for (n = 0; n < image->Width; n++) {
            theta = (-2) * PI * (j * n) / image->Width;
            Rg[i] [ j] += Rt[i][n] * cos(theta) - It[i][n] * sin(theta);
            Ig[i] [ j] += Rt[i][n] * sin(theta) + It[i][n] * cos(theta);
        }
        res = sqrt(pow(Rg[i][j], 2) + pow(Ig[i][j], 2))
        tempout[i][j] = res;
    }
}
```

2. IDFT Using Phase Angle

Input: Result of DFT including two array that stores the real Rg and imaginary Ig part of the image

Output: an unsigned char array that stores the output image data

$$f(m,n) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(u,v)\, e^{\frac{j2\pi um}{M}} e^{\frac{j2\pi uv}{N}}$$

Calculate the phase spectrum $\varphi(u,v) = \arctan\left(\frac{I(u,v)}{R(u,v)}\right)$.

Creat a zero image t, whose real part is Rt, imaginary part is It.

Only use phase information reconstruction means $|F(u,v)| = |F(u,v)|e^{j\varphi(u,v)}$, let $|F(u,v)| = 1$, we can get the phase angle information.

```
for (i = 0; i < inimage.Height; i++) {
    for (j = 0; j < inimage.Width; j++) {
        mag= sqrt(pow(Ig[i][ j], 2) + pow(Rg[i][ j], 2));
        Rg[i][ j] = Rg[i][ j] / mag;
        Ig[i][ j] = Ig[i][ j] / mag;            }
}
for (i = 0; i < inimage.Height; i++) {
    for (j = 0; j < inimage.Width; j++) {
        for (m = 0; m < image.Height; m++) {
            theta = 2 * PI * (i * m / image->Height);
            Rt[i ][ j] +=Rg[m][j] * cos(theta) - Ig[m][j] * sin(theta);
            It[i ][ j] += Ig[m][j] * sin(theta) + Ig[m][j] * cos(theta);
        }
    }
}

for (i = 0; i < inimage.Height; i++) {
    for (j = 0; j < inimage.Width; j++) {
        real = 0;
        imaginary = 0;
        for (n = 0; n < image->Width; n++) {
            theta = 2 * PI * (j * n) / image.Width;
            real+= Rt[i ][n] * cos(theta) - It[i ][n] * sin(theta);
            imaginary += Rt[i ][n] * sin(theta) + It[i ][n] * cos(theta);
        }
        res =sqrt(pow(imaginary, 2) + pow(real, 2));
        outimage [i ][ j] = res;
return outimage
```

3. IDFT using magnitude spectrum

Input: Result of DFT including two array that stores the real and imaginary part of the image

Output: an unsigned char array that stores the output image data

$$f(m,n) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(u,v)\, e^{\frac{j2\pi um}{M}} e^{\frac{j2\pi uv}{N}}$$

Calculate the phase spectrum $F(u,v) = (I(u,v)^2 + R(u,v)^2)^{-1}$

Shift the image $F(u,v) = (-1)^{u+v}F(u,v)$

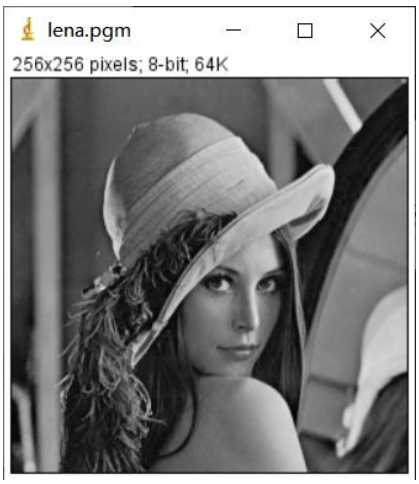
Creat a zero image t, whose real part is Rt, imaginary part is It.

Only use phase information reconstruction means $|F(u,v)| = |F(u,v)|e^{j\varphi(u,v)}$, let phase angle =0 , we can get the magnitude information.

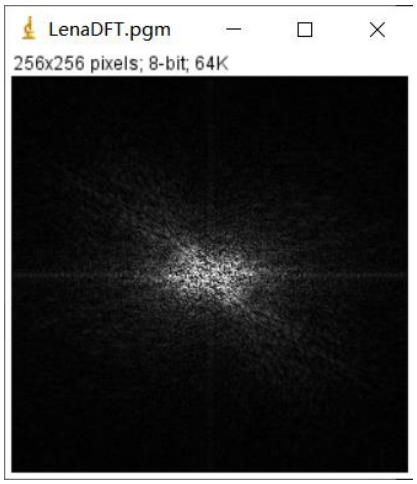
```
for (i = 0; i < inimage.Height; i++) {
    for (j = 0; j < inimage.Width; j++) {
        mag=sqrt(pow(Rg[i ][ j], 2) + pow(Ig[i ][ j], 2))/size;
        Rg[i ][ j] = mag;
        Ig[i ][ j] = 0;          }
}
for (i = 0; i < inimage.Height; i++) {
    for (j = 0; j < inimage.Width; j++) {
        for (m = 0; m < image.Height; m++) {
            theta = 2 * PI * (i * m / image->Height);
            Rt[i ][ j] +=Rg[m][j] * cos(theta) - Ig[m][j] * sin(theta);
            It[i ][ j] += Ig[m][j] * sin(theta) + Ig[m][j] * cos(theta);
        }
    }
}

for (i = 0; i < inimage.Height; i++) {
    for (j = 0; j < inimage.Width; j++) {
        real = 0;
        imaginary = 0;
        for (n = 0; n < image->Width; n++) {
            theta = 2 * PI * (j * n) / image.Width;
            real+= Rt[i ][n] * cos(theta) - It[i ][n] * sin(theta);
            imaginary += Rt[i ][n] * sin(theta) + It[i ][n] * cos(theta);
        }
        res =sqrt(pow(imaginary, 2) + pow(real, 2));
        outimage [i ][ j] = res;
return outimage
```

Results (compare the results with the original image):

1. Fourier Transform

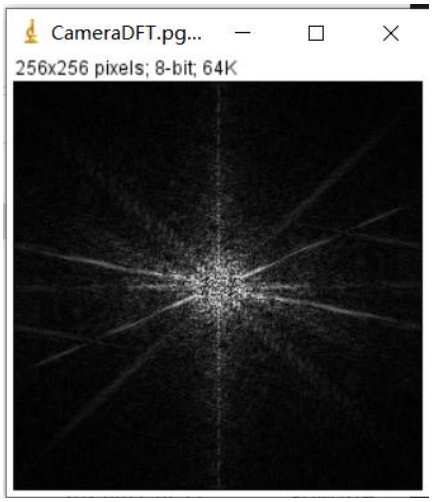


Source



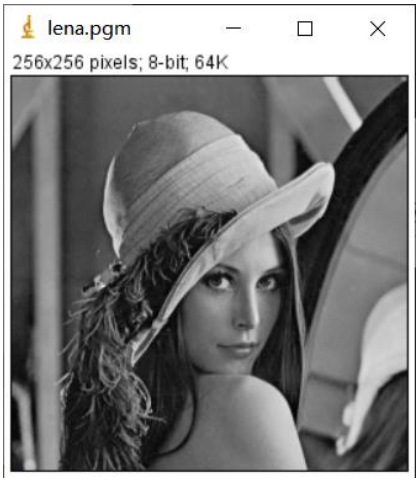
After

Source

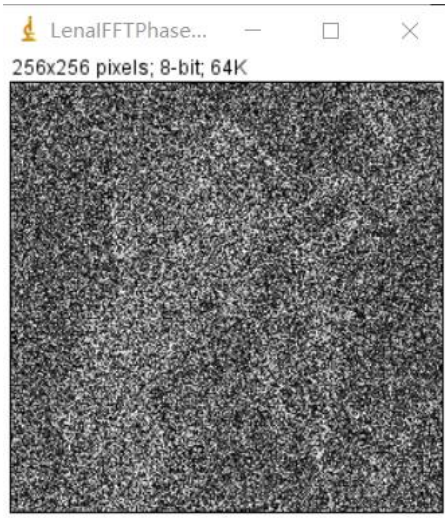


After

2. IDFT Using Phase spectrum

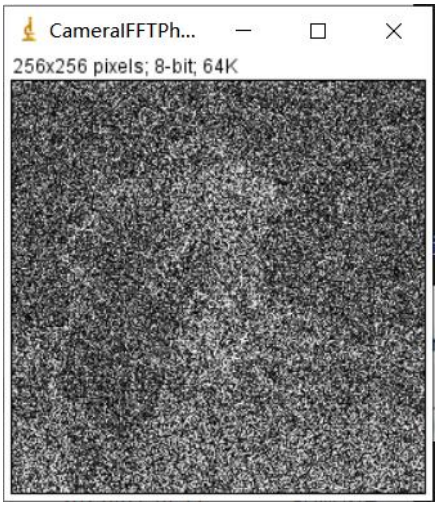


Source



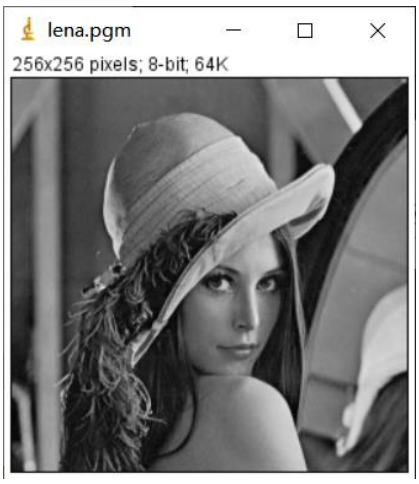
After

Source

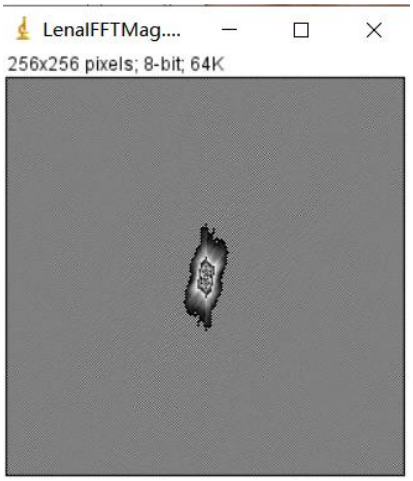


After

3. IDFT Using Magnitude spectrum

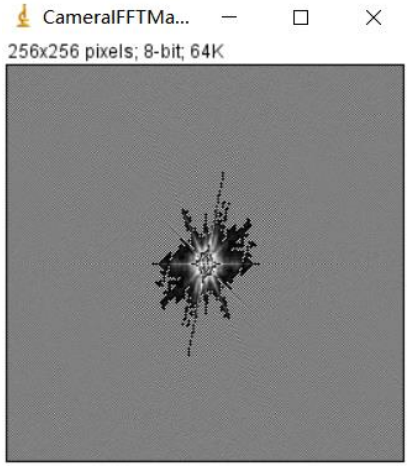


Source



After

Source



After

Discussion:

1. After shifting the image, the frequency spectrum can move to the central of the image
2. The original image cannot be reconstructed from the amplitude spectrum, which also verifies that the amplitude spectrum only contains the gray information of the image, and the phase spectrum plays a decisive role in the image content.

3. The phase spectrum is used to reconstruct the contour information of the image, so it is easy to verify that the phase spectrum of the image stores the position information of the image. However, due to the lack of amplitude spectrum, the reconstructed image lacks the variation in pixel value.

Codes:

1. Fourier Transform

```
Image* DFTSepatate(Image* image, float* Rg, float* Ig) {
    unsigned char* tempin, * tempout;
    int size, i, j, m, n;
    float real, imaginary, sum;
    float theta, res;
    Image* outimage;
    float* Rt = (float*)calloc(image->Width * image->Height, sizeof(float));
    float* It = (float*)calloc(image->Width * image->Height, sizeof(float));
    outimage = CreateNewImage(image, "#Discrete Fourier Transform",image->Width,image->Height);
    tempin = image->data;
    tempout = outimage->data;

    for (i = 0; i < image->Height; i++) {  // Shift image
        for (j = 0; j < image->Height; j++) {
            tempin[image->Width * i + j] = pow((-1), (i + j)) * tempin[image->Width * i + j];
        }
    }

    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            for (m = 0; m < image->Height; m++) {
                theta = (float)(-2) * PI * ((float)i * (float)m) / (float)(image->Height);
                Rt[i* image->Width  + j] += (float)tempin[m*image->Width  + j] * cos(theta);
                It[i* image->Width  + j] += (float)tempin[m*image->Width  + j] * sin(theta);
                //printf("%f\n", theta);
            }
        }
    }

    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            for (n = 0; n < image->Width; n++) {
                theta = (float)(-2) * PI * ((float)j * (float)n) / (float)image->Width;
                Rg[i *image->Width  + j] += Rt[image->Width * i + n] * cos(theta) - It[image->Width * i + n] * sin(theta);
                Ig[i *image->Width  + j] += Rt[image->Width * i + n] * sin(theta) + It[image->Width * i + n] * cos(theta);
            }

            res = sqrt(pow(Rg[i * image->Width + j], 2) + pow(Ig[i * image->Width + j], 2)) / sqrt(image->Width*image->Height);
            tempout[image->Width * i + j] = res;
        }
    }

    free(Rt);
    free(It);

    return(outimage);
}
```

2. IDFT Using Phase spectrum

```
Image* IDFTPhaseAngle(Image* image, float* Rg, float* Ig) {
    unsigned char* tempin, * tempout,*temp;
    int size, i, j, m, n;
    float res;
    float real, imaginary, sum;
    float theta,mag;
    Image* outimage, *tempimage;
    outimage = CreateNewImage(image, "#Discrete Fourier Transform", image->Width, image->Height);
    tempout = outimage->data;

    float* Rt = (float*)calloc(image->Width * image->Height, sizeof(float));
    float* It = (float*)calloc(image->Width * image->Height, sizeof(float));

    int imsize = image->Height * image->Width;
    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            mag= sqrt(pow(Ig[i * image->Width + j], 2) + pow(Rg[i * image->Width + j], 2));
            Rg[i * image->Width + j] = Rg[i * image->Width + j] / mag;
            Ig[i * image->Width + j] = Ig[i * image->Width + j] / mag;
        }
    }

    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            for (m = 0; m < image->Height; m++) {
                theta = 2 * PI * (float)((float)i * (float)m / (float)image->Height);
                Rt[i * image->Width + j] += Rg[m * image->Width + j] * cos(theta) - Ig[m * image->Width + j] * sin(theta);
                It[i * image->Width + j] += Rg[m * image->Width + j] * sin(theta) + Ig[m * image->Width + j] * cos(theta);
            }
        }
    }

    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            real = 0;
            imaginary = 0;
            for (n = 0; n < image->Width; n++) {
                theta = 2 * PI * (float)((float)j * (float)n / (float)image->Width);
                real+= Rt[i * image->Width + n] * cos(theta) - It[i * image->Width + n] * sin(theta);
                imaginary += Rt[i * image->Width + n] * sin(theta) + It[i * image->Width + n] * cos(theta);
            }
            res = sqrt(pow(imaginary, 2) + pow(real, 2));
            tempout[image->Width * i + j] = res;

        }
    }

    free(Rt);
    free(It);
    return(outimage);
}
```

3. IDFT Using Magnitude spectrum

```
Image* IDFTMagnitude(Image* image, float* Rg, float* Ig) {
    unsigned char* tempin, * tempout, * temp;
    int size, i, j, m, n;
    float res;
    float real, imaginary, sum;
    float theta;
    Image* outimage, * tempimage;
    outimage = CreateNewImage(image, "#Discrete Fourier Transform", image->Width, image->Height);
    tempout = outimage->data;

    float* Rt = (float*)calloc(image->Width * image->Height, sizeof(float));
    float* It = (float*)calloc(image->Width * image->Height, sizeof(float));
    float* mag= (float*)calloc(image->Width * image->Height, sizeof(float));
    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            mag[i * image->Width + j] = sqrt(pow(Ig[i * image->Width + j], 2) + pow(Rg[i * image->Width + j], 2))/(float)(image->Width*image->Height);
            Rg[i * image->Width + j] = mag[i * image->Width + j];
            Ig[i * image->Width + j] = 0;

        }
    }
    for (i = 0; i < image->Height; i++) {  // Shift image
        for (j = 0; j < image->Width; j++) {
            Rg[i * image->Width + j] = Rg[i * image->Width + j] * pow((-1), (i + j));
        }
    }

    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            for (m = 0; m < image->Height; m++) {
                theta = 2 * PI * ((float)i * (float)m / (float)image->Height);
                Rt[i * image->Width + j] += Rg[m * image->Width + j] * cos(theta);
                It[i * image->Width + j] += Rg[m * image->Width + j] * sin(theta);
            }

        }
    }

    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            real = 0;
            imaginary = 0;
            for (n = 0; n < image->Width; n++) {
                theta = 2 * PI * (float)((float)j * (float)n / (float)image->Width);
                real += Rt[i * image->Width + n] * cos(theta) - It[i * image->Width + n] * sin(theta);
                imaginary += Rt[i * image->Width + n] * sin(theta) + It[i * image->Width + n] * cos(theta);
            }
            res = sqrt(pow(imaginary, 2) + pow(real, 2));
            tempout[image->Width * i + j] = res;
        }
    }
    free(Rt);
    free(It);
    return(outimage);
}
```