

## 5. Fourier Transform and Reconstruction

### 1. 2D Fourier Transform (lena, bridge):

- DFT:

#### Algorithm:

The two-dimensional discrete Fourier transform formula is as follows:

$$F(u, v) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M+vy/N)}$$

where  $f(x, y)$  is the original pixel, and  $F(u, v)$  is the result after Fourier transform. According to Euler's formula  $C = |C|e^{j\theta}$ , each value of  $F(u, v)$  is a complex number, which is calculated by the real part  $R$  and Imaginary part  $I$ .

The formula for the Fourier spectrum:

$$F(u, v) = |F(u, v)|^2 = [R^2(u, v) + I^2(u, v)]$$

After the transformation, we want the four frequencies to meet at the center of the image, which requires a Fourier translation:

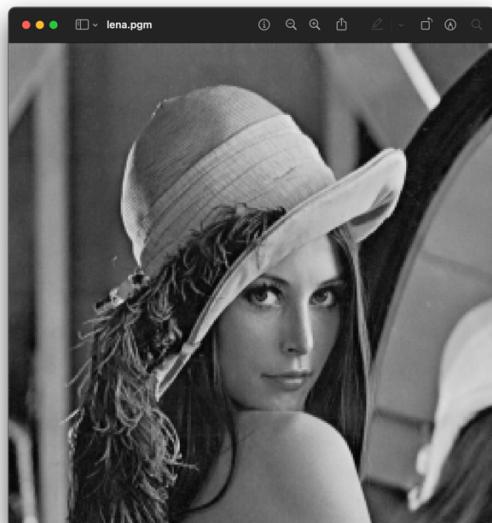
$$f(x, y)(-1)^{x+y} \Leftrightarrow F\left(u - \frac{M}{2}, v - \frac{N}{2}\right)$$

So multiplying each  $f(x, y)$  term by  $(-1)^{x+y}$  achieves the purpose

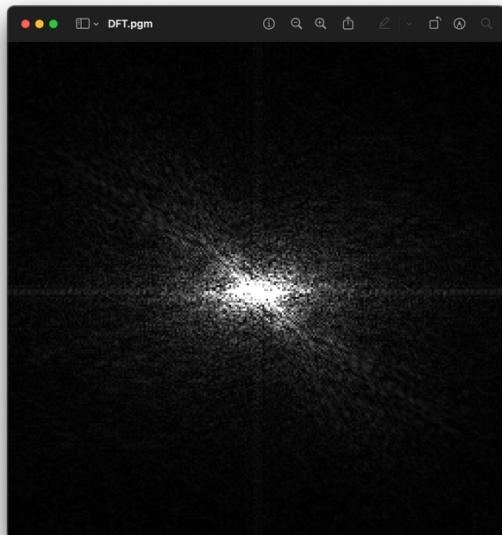
#### Results (including pictures):

Process result of "lena.pgm":

Source Image:

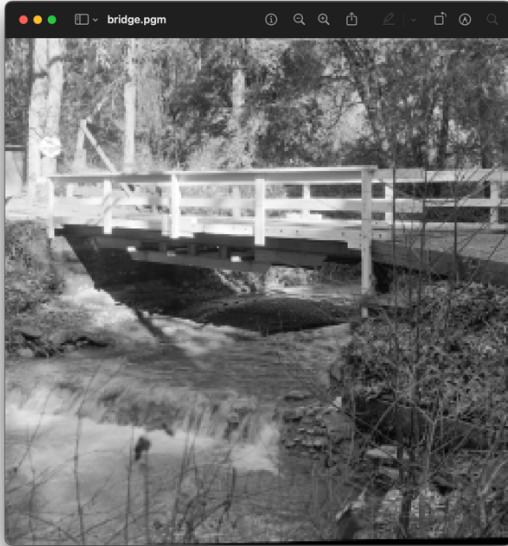


Result of Fourier spectrum:

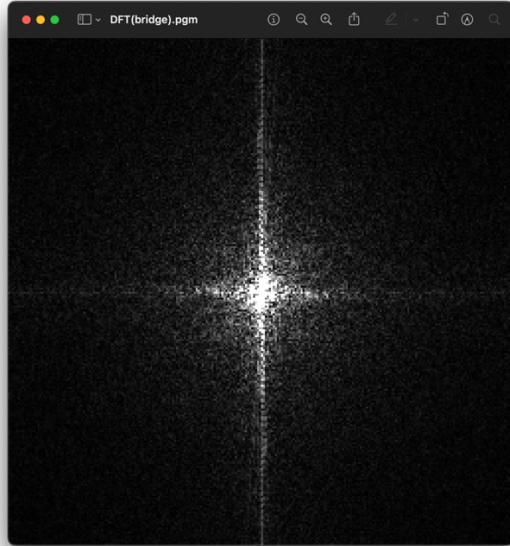


Process result of "bridge.pgm":

Source Image:



Result of Fourier spectrum:



### Discussion:

The frequency of an image is an indicator that characterizes the intensity of grayscale changes in an image, and is the gradient of grayscale in plane space. The right image is the corresponding spectrum, the middle part is very bright and the gray value is very high. This means that the lower frequency waves will have very large intensities, while the higher frequency waves will have very small intensities. However, in the process of testing, it is found that the execution speed is really very slow, and the time complexity of the algorithm is  $O(n^4)$ .

### Codes:

```
48 // Algorithms Code:
49 Image *DFT(Image *image, float *real_array, float *imaginary_array) {
50     unsigned char *tempin, *tempout;
51     float real, imaginary;
52     Image *outimage;
53     outimage = CreateNewImage(image, (char*)"#testing function");
54     tempin = image->data;
55     tempout = outimage->data;
56
57     printf("DFT algorithm is executing...\n");
58     for(int i = 0; i < image->Height; i++) {
59         for(int j = 0; j < image->Width; j++) {
60             real = 0;
61             imaginary = 0;
62
63             for(int m = 0; m < image->Height; m++) {
64                 for(int n = 0; n < image->Width; n++) {
65                     float temp = (float)i * m / (float)image->Height + (float)j * n / (float)image->Width;
66                     int offset = (m + n) % 2 == 0 ? 1 : -1;
67                     real += tempin[image->Width * m + n] * cos(-2 * pi * temp) * offset;
68                     imaginary += tempin[image->Width * m + n] * sin(-2 * pi * temp) * offset;
69                 }
70             }
71             real_array[image->Width * i + j] = real;
72             imaginary_array[image->Width * i + j] = imaginary;
73
74             int temp = (int)(sqrt(real*real + imaginary*imaginary) / sqrt((float)image->Height*(float)image->Width));
75             if(temp < 0) temp = 0;
76             if(temp > 255) temp = 255;
77             tempout[image->Width * i + j] = temp;
78         }
79     }
80     printf("DFT Finished!\n");
81     return (outimage);
82 }
```

- iDFT:

**Algorithm:**

Inverse transform the image using the real part  $R$  and imaginary part  $I$  obtained from the DFT.  
The formula of iDFT is:

$$f(x, y) = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M+vy/N)}$$

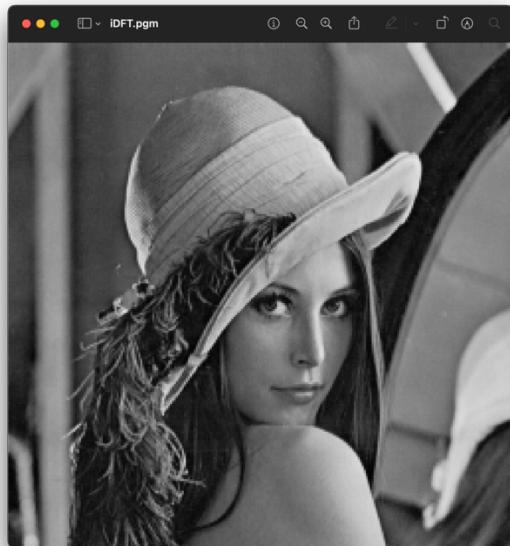
**Results (including pictures):**

Process result of "lena.pgm":

Source Image:



iDFT reconstruct:



Process result of "bridge.pgm":

Source Image:



iDFT reconstruct:

**Discussion:**

It has been verified that the original image can be restored after the image is Fourier transformed and then inversely transformed.

**Codes:**

```

84 Image *iDFT(Image *image, float *real_array, float *imaginary_array) {
85     unsigned char *tempin, *tempout;
86     float real;
87     Image *outimage;
88     outimage = CreateNewImage(image, (char*)"#testing function");
89     tempin = image->data;
90     tempout = outimage->data;
91
92     printf("iDFT algorithm is executing...\n");
93     for(int i = 0; i < image->Height; i++) {
94         for(int j = 0; j < image->Width; j++) {
95             real = 0;
96
97             for (int m = 0; m < image->Height; m++){
98                 for (int n = 0; n < image->Width; n++){
99                     float temp = (float)i * m / (float)image->Height + (float)j * n / (float)image->Width;
100                    real += real_array[image->Width * m + n] * cos(2 * pi * temp) -
101                        imaginary_array[image->Width * m + n] * sin(2 * pi * temp);
102                }
103            }
104            int offset = (i + j) % 2 == 0 ? 1 : -1;
105            int temp = (int)(real / ((float)image->Height*(float)image->Width) * offset);
106            if(temp < 0) temp = 0;
107            if(temp > 255) temp = 255;
108            tempout[image->Width * i + j] = temp;
109        }
110    printf("iDFT Finished!\n");
111    return (outimage);
112 }
```

## 2. Reconstruction only using the phase angle from 2D DFT (lena):

### Algorithm:

According to the formulas of iDFT and phase angle:

$$\phi(u, v) = \arctan \left[ \frac{I(u, v)}{R(u, v)} \right]$$

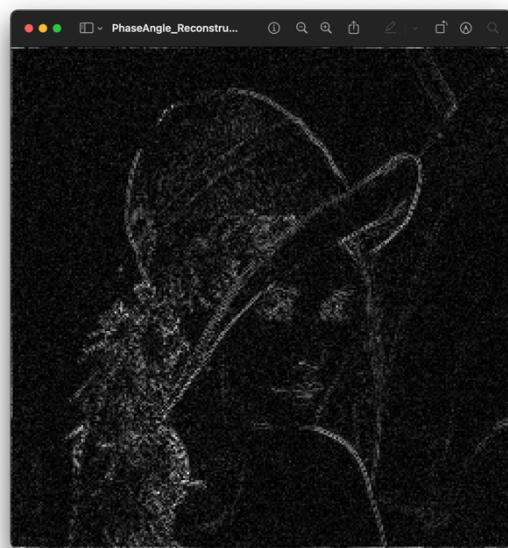
$$f(x, y) = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M+vy/N)}$$

If we want to reconstruct the image only using the phase angle, we need to set  $F(u, v)$  to 1.

### Results (including pictures):

Process result of "lena.pgm":

Source Image:



Result after phase angle reconstruct:

**Discussion:**

As can be seen from the features in the image, the phase information correlates with key shape properties in the image, although the grayscale information has been lost.

**Codes:**

```

114 Image *Reconstruct_phaseAngle(Image *image, float *real_array, float *imaginary_array) {
115     unsigned char *tempin, *tempout;
116     float *tempArray;
117     float real, angle, size = image->Width * image->Height;
118     int max = 0, min = 10000; // used to normalization
119     Image *outimage;
120
121     outimage = CreateNewImage(image, (char*)"#testing function");
122     tempin = image->data;
123     tempout = outimage->data;
124     tempArray = (float*)malloc(size * sizeof(float));
125
126     printf("Phase Angle Reconstruction is executing...\n");
127     for(int i = 0; i < image->Height; i++) {
128         for(int j = 0; j < image->Width; j++) {
129             real = 0;
130
131             for (int m = 0; m < image->Height; m++){
132                 for (int n = 0; n < image->Width; n++){
133                     float temp = (float)i * m / (float)image->Height + (float)j * n / (float)image->Width;
134                     angle = atan2(imaginary_array[image->Width * m + n], real_array[image->Width * m + n]);
135                     real += cos(angle) * cos(2 * pi * temp) - sin(angle) * sin(2 * pi * temp);
136                 }
137             }
138             int temp = round(fabs(real));
139             tempArray[image->Width * i + j] = temp;
140             if(temp > max) max = temp;
141             if(temp < min) min = temp;
142         }
143     }
144     // Normalization:
145     for(int i = 0; i < size; i++) {
146         float temp = ((float)tempArray[i] - (float)min) / ((float)max - (float)min);
147         tempout[i] = (int)(temp * 256);
148     }
149     printf("Phase Angle Reconstruction Finished!\n");
150     return (outimage);
151 }
```

### 3. Reconstruction only using the magnitude from 2D DFT (lena):

#### Algorithm:

According to the formulas of iDFT and amplitude:

$$|F(u, v)| = [R^2(u, v) + I^2(u, v)]^{\frac{1}{2}}$$

$$f(x, y) = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M+vy/N)}$$

If we want to reconstruct the image only using the magnitude, we need to set *exponent* to 0, that is,  $e^{j2\pi(ux/M+vy/N)} = 1$ .

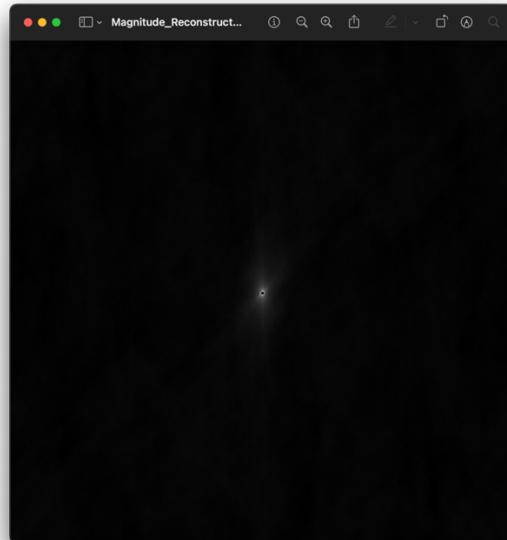
#### Results (including pictures):

Process result of "lena.pgm":

Source Image:



Result after magnitude reconstruct:



#### Discussion:

The resulting image uses only the spectrum obtained by inverse Fourier transform, which also means setting the phase angle to 0. So the output contains only grayscale information, and the DC term dominates. There is no shape information in the image because the phase has been set to zero.

**Code:**

```
153 Image *Reconstruct_magnitude(Image *image, float *real_array, float *imaginary_array) {
154     unsigned char *tempin, *tempout;
155     float *tempArray;
156     float real, imaginary, size = image->Width * image->Height;
157     int max = 0, min = 10000; // used to normalization
158
159     Image *outimage;
160     outimage = CreateNewImage(image, (char*)"#testing function");
161     tempin = image->data;
162     tempout = outimage->data;
163     tempArray = (float*)malloc(size * sizeof(float));
164
165     printf("Magnitude Reconstruction is executing...\n");
166     for(int i = 0; i < image->Height; i++) {
167         for(int j = 0; j < image->Width; j++) {
168             real = 0;
169             imaginary = 0;
170
171             for (int m = 0; m < image->Height; m++){
172                 for (int n = 0; n < image->Width; n++){
173                     float temp = (float)i * m / (float)image->Height + (float)j * n / (float)image->Width;
174                     float mag = sqrt(pow(real_array[image->Width * m + n], 2) + pow(imaginary_array[image->Width * m + n], 2)) /
175                         sqrt(size);
176                     real += mag * cos(2 * pi * temp) * pow(-1, (float)m + (float)n);
177                     imaginary += mag * sin(2 * pi * temp) * pow(-1, (float)m + (float)n);
178                 }
179                 int temp = (int)sqrt(pow(real, 2) + pow(imaginary, 2));
180                 tempArray[image->Width * i + j] = temp;
181                 if(temp > max) max = temp;
182                 if(temp < min) min = temp;
183             }
184         }
185         // Normalization:
186         for(int i = 0; i < size; i++) {
187             float temp = ((float)tempArray[i] - (float)min) / ((float)max - (float)min);
188             tempout[i] = (int)(temp * 256);
189         }
190         printf("Magnitude Reconstruction Finished!\n");
191     }
192 }
```