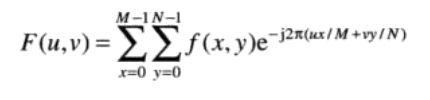
**Lab 5**

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

1. **DFT (lena, image from the textbook):**

**Algorithm:**



The above is the basic function of discrete Fourier Transformation.

We use the Euler function to convert the logarithm part to real part and imaginary part.

In this process, we need to use to multiply the image data to centralize the spectrum.

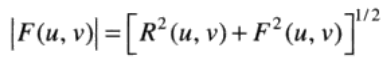
temp = -2 \* pi \* (i \* x / height + j \* y / width);

move = (x + y) % 2 == 0 ? 1 : -1

re += cos(temp) \* (\*(tempin + width \* x + y))\*move

im += sin(temp) \* (\*(tempin + width \* x + y))\*move

Then, we need to use the spectrum to show the image because we can’t use the complex to represent the image data. The spectrum function is as follow:



sqrt(re \* re + im \* im);

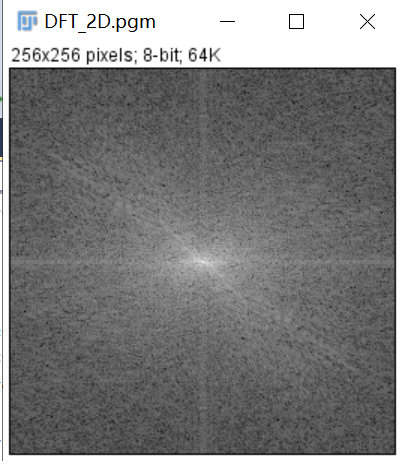
Finally, we can use (1+log( | F(u,v) | ) ) to improve details.

\*(tempout + i \* width + j) = 1 + log((sqrt(re\*re + im\*im)))

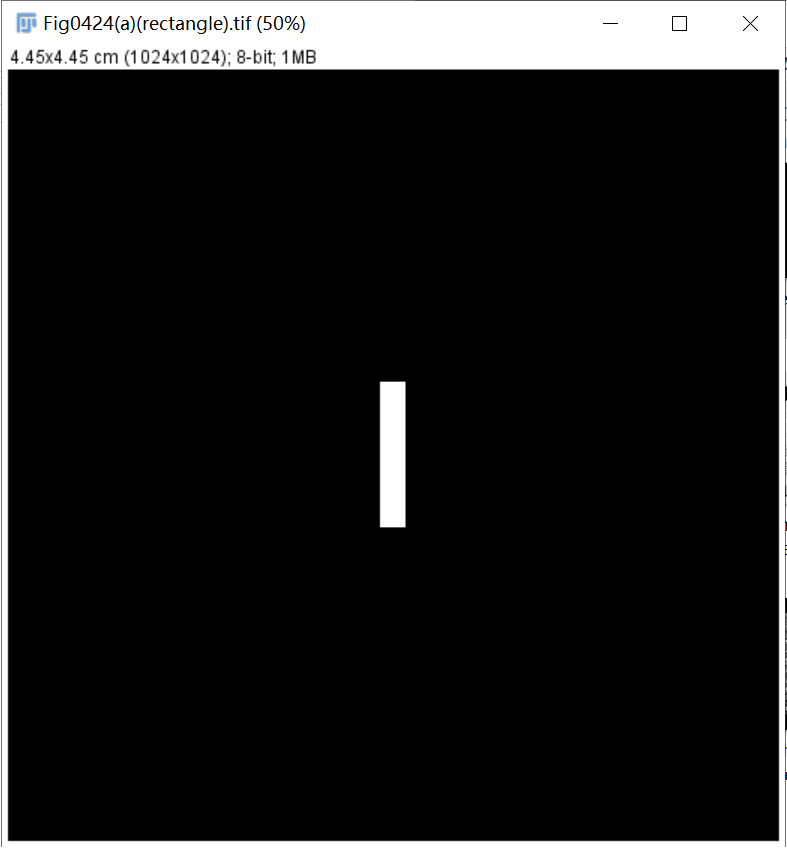
Meanwhile, remember to normalize all the image data to the interval [0,255]

**Results (including pictures):**

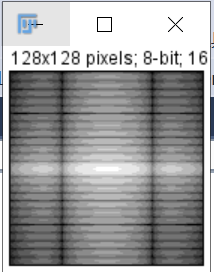
Source: result:

Source:



result:



**Discussion:**

Actually, it is enough to show the result just using basic function of DFT and spectrum. All other steps are to make the final result more beautiful. If we don’t centralize the spectrum, the high value will show in the four corners (In the image, it appears as the brightest part). After the centralization, the brightest part will appear in the center.

Because we use multiple loops, the running time is very long. We can reduce the size of the picture to get the results quickly.

Because the picture is reduced, the details become different, and the results are different from those in the book

**Codes:**

Image \*DFT2(Image\* image) {

int width, height;

width = image->Width;

height = image->Height;

Image\* outimage;

unsigned char\* tempin, \*tempout;

const char\* comment = "#testing DFT2";

outimage = CreateNewImage(image, comment);

tempin = image->data;

tempout = outimage->data;

double\* re1, \*im1, pi = acos(-1.0);

re1 = (double\*)malloc(width\*height \* sizeof(double));

im1 = (double\*)malloc(width\*height \* sizeof(double));

int i, j, x, y,move;

double temp, re, im;

for (i = 0; i < height; i++) {

for (j = 0; j < width; j++) {

re = 0;

im = 0;

for (x = 0; x < height; x++) {

for (y = 0; y < width; y++) {

temp = -2 \* pi \* ((double)i \* (double)x / (double)height + (double)j \* (double)y / (double)width);

move = (x + y) % 2 == 0 ? 1 : -1;

re += cos(temp) \* (double)(\*(tempin + width \* x + y))\*move;

im += sin(temp) \* (double)(\*(tempin + width \* x + y))\*move;

}

}//DFT

if (re1 && im1) {//Deal with C6011

\*(re1 + i \* width + j) = re;

\*(im1 + i \* width + j) = im;

}

\*(tempout + i \* width + j) = (double)(1 + log((sqrt(re\*re + im\*im))));

}

}

double max = tempout[0];

double min = tempout[0];

for (i = 0; i < height; i++)

{

for (j = 0; j < width; j++)

{

if (tempout[i\*width + j] >= max){

max = tempout[i\*width + j];

}

if (tempout[i\*width + j] <= min){

min = tempout[i\*width + j];

}

}

}

for (i = 0; i < height; i++)

{

for (j = 0; j < width; j++)

{

tempout[i\*width + j] = (double)(tempout[i\*width + j] - min) \* 255 / (max - min);

}

}

free(re1);

free(im1);

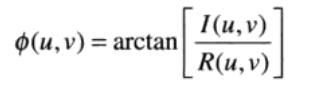
return(outimage);

}

1. **Phase angle(lena):**

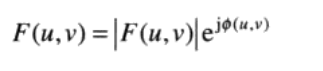
**Algorithm:**

First, we use the DFT to get the real and imaginary part.



atan2(im, re)

Then, we use the above function to get the phase angle.

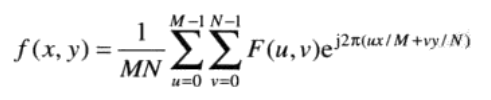


After the angle is brought into the above formula, the image processed by phase angle is obtained.

In this situation, | F(u,v) | = 1.

re = cos((\*(tempout + width \* i + j)));

im = sin((\*(tempout + width \* i + j)));

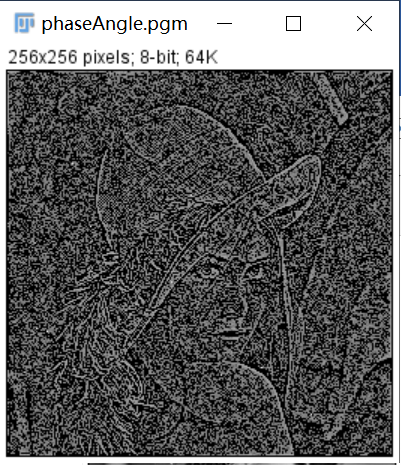


Finally, we use the IDFT to reconstruct the image.

re += cos(temp) \* (\*(re1 + width \* x + y)) - sin(temp) \* (\*(im1 + width \* x + y));

**Results (including pictures):**

Source: result:

**Discussion:**

The image is just reconstructed by the phase information, so the gray information has lost.

However, we can still distinguish the approximate shape of the original image.

Interestingly, after the above algorithm processing, the image is like a noise image.

I can roughly distinguish the shape of the original image after using 1+20\*log().

Different logarithmic multiples can get different results. After my test, 10-30 can distinguish the shape. I only tested some data.

Because we use multiple loops, the running time is very long. We can reduce the size of the picture to get the results quickly.

**Codes:**

Image \*phaseAngle(Image\* image) {

int width, height;

width = image->Width;

height = image->Height;

Image\* outimage;

unsigned char\* tempin, \*tempout;

const char\* comment = "#testing phase angle";

outimage = CreateNewImage(image, comment);

tempin = image->data;

tempout = outimage->data;

//Store IDFT data

double re, im, temp,pi = acos(-1.0);

double\* re1, \*im1;

re1 = (double\*)malloc(width\*height \* sizeof(double));

im1 = (double\*)malloc(width\*height \* sizeof(double));

int i, j, x, y;

for (i = 0; i < height; i++) {

for (j = 0; j < width; j++) {

re = 0;

im = 0;

for (x = 0; x < height; x++) {

for (y = 0; y < width; y++) {

temp = -2 \* pi \* ((double)i \* (double)x / (double)height + (double)j \* (double)y / (double)width);

re += cos(temp) \* (double)(\*(tempin + width \* x + y));

im += sin(temp) \* (double)(\*(tempin + width \* x + y));

}

}//DFT

if (re1 && im1) {//Deal with C6011

\*(re1 + i \* width + j) = re;

\*(im1 + i \* width + j) = im;

}

\*(tempout + i \* width + j) = atan2(im, re);//phase angle

}

}

for (i = 0; i < (int)height; i++) {

for (j = 0; j < (int)width; j++) {

re = cos((double)(\*(tempout + width \* i + j)));

im = sin((double)(\*(tempout + width \* i + j)));

if (re1 && im1) {//Deal with C6011

\*(re1 + i \* width + j) = re;

\*(im1 + i \* width + j) = im;

}

}

}

for (i = 0; i < height; i++) {

for (j = 0; j < width; j++) {

re = 0;

im = 0;

for (x = 0; x < height; x++) {

for (y = 0; y < width; y++) {

temp = 2 \* pi \* ((double)i \* (double)x / (double)height + (double)j \* (double)y / (double)width);

if (re1 && im1) {//Deal with C6011

re += cos(temp) \* (double)(\*(re1 + width \* x + y)) - sin(temp) \* (double)(\*(im1 + width \* x + y));

}

}

}//IDFT

\*(tempout + i \* width + j) = 1 + 20 \* log(re / (width\*height));

}

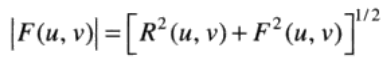
}return(outimage);

}

1. **magnitude (lena):**

**Algorithm:**

First, we use the DFT to get the real and imaginary part.



Then, we use the above function to get the spectrum.

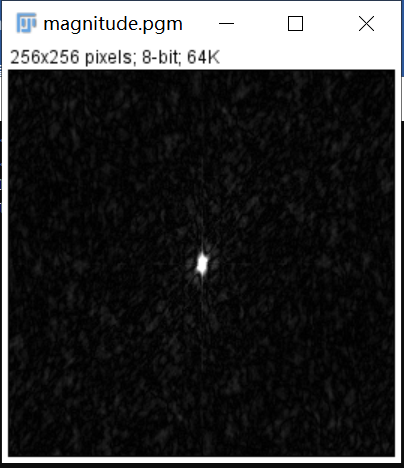
In this situation, the phase angel equals to 0, so F(u,v)= | F(u,v) |.

Finally, we use the IDFT to reconstruct the image.

Also remember to centralize the image.

**Results (including pictures):**

Source: result:

**Discussion:**

The final result only contains gray information, and the dc term is dominant. There is no shape information in the image because the phase has been set to zero.

After IDFT, I get the results that are similar to those in the book by using the spectrum and divide it by radical sign of the image size. Otherwise, the result will be different.

Because we use multiple loops, the running time is very long. We can reduce the size of the picture to get the results quickly.

**Codes:**

Image \*magnitude(Image\* image) {

int width, height;

width = image->Width;

height = image->Height;

Image\* outimage;

unsigned char\* tempin, \*tempout;

const char\* comment = "#testing magnitude";

outimage = CreateNewImage(image, comment);

tempin = image->data;

tempout = outimage->data;

double\* re1, \*im1;

re1 = (double\*)malloc(width\*height \* sizeof(double));

im1 = (double\*)malloc(width\*height \* sizeof(double));

int i, j, x, y,move;

double temp, re, im, pi = acos(-1.0);;

for (i = 0; i < height; i++) {

for (j = 0; j < width; j++) {

re = 0;

im = 0;

for (x = 0; x < height; x++) {

for (y = 0; y < width; y++) {

temp = -2 \* pi \* ((double)i \* (double)x / (double)height + (double)j \* (double)y / (double)width);

move = (x + y) % 2 == 0 ? 1 : -1;

re += cos(temp) \* (double)(\*(tempin + width \* x + y))\*move;

im += sin(temp) \* (double)(\*(tempin + width \* x + y))\*move;

}

}//DFT

if (re1 && im1) {//Deal with C6011

\*(re1 + i \* width + j) = re;

\*(im1 + i \* width + j) = im;

}

\*(tempout + i \* width + j) = temp = sqrt(re \* re + im \* im) / sqrt((double)(width\*height));;//magnitude

}

}

for (i = 0; i < height; i++) {

for (j = 0; j < width; j++) {

re = (double)(\*(tempout + width \* i + j));

im = 0;

if (re1 && im1) {//Deal with C6011

\*(re1 + i \* width + j) = re;

\*(im1 + i \* width + j) = im;

}

}

}

for (i = 0; i < height; i++) {

for (j = 0; j < width; j++) {

re = 0;

im = 0;

for (x = 0; x < height; x++) {

for (y = 0; y < width; y++) {

temp = 2 \* pi \* ((double)i \* (double)x / (double)height + (double)j \* (double)y / (double)width);

if (re1 && im1) {//Deal with C6011

move = (x + y) % 2 == 0 ? 1 : -1;

re += cos(temp) \* (double)(\*(re1 + width \* x + y))\*move;

im += sin(temp) \* (double)(\*(im1 + width \* x + y))\*move;

}

}

}//IDFT

temp = sqrt(re \* re + im \* im) / sqrt((double)(width\*height));

if (temp > 255) {

temp = 255;

}

if (temp < 0) {

temp = 0;

}

\*(tempout + i \* width + j) = temp;

}

}

free(re1);

free(im1);

return(outimage);

}