暨南大学本科实验报告专用纸

课程名称　　数字图像处理　　　　成绩评定

实验项目名称　　图像的傅立叶变换　　　指导教师　刘晓翔

实验项目编号　03　实验项目类型　综合型　实验地点　三楼机房

学生姓名　　　赵俊文　　　　学号　　2022104002

学院　　智能科学与工程学院　　系　　　专业　　人工智能

实验时间 2024　年 10 月 7 日　上 午～ 10 月 28 日　上 午

（一）　实验目的

①了解图像变换的意义和手段；②熟悉傅里叶变换的公式和基本性质；③掌握基本傅立叶变换（FT）及其反变换的编程方法；④掌握快速傅立叶变换（FFT）及其反变换的编程方法。

（二）　实验内容和要求

利用 Visual C++6.0 软件开发工具编写程序，根据 FT、FFT 算法生成 256 灰度图像频谱图，根据 FT、FFT 反变换实现频域数据到原图像的恢复，程序执行结果正确。

（三）　主要仪器设备

仪器：计算机

实验环境：　Windows XP + Visual C++6.0

（四）　实验步骤（附代码）与调试

1.FT&IFT

a.根据 FT 算法以及反变换计算方法，在 bmp 文件中添加两个函数分别实现傅里叶变换和反变换计算方法，然后再添加两个函数对图像进行傅里叶变换和反变换。代码如下

①定义变换后的图像指针

```
BITMAPINFO* lpDIB_FT = NULL;
```

BITMAPINFO* lpDIB_IFT = NULL;

complex <double> *gFD =NULL;

②一维傅里叶变换和反变换的计算公式代码实现

```
//一维傅里叶变换
void FT(complex<double>* TD,complex<double>* FD,int m)
{
    int x,u;
    double angle;
    for (u=0;u<m;u++){
        FD[u]=0;
        for(x=0;x<m;x++){
            angle=-2*PI*u*x/m;
            FD[u]+=TD[x]*complex<double>(cos(angle),sin(angle));
        }
        FD[u]/=m;
    }

}

//反变换
void IFT(complex<double>* FD,complex<double>* TD,int m)
{
    int x,u;
    double angle;
    for (x=0;x<m;x++){
        TD[x]=0;
        for(u=0;u<m;u++){
            angle=2*PI*u*x/m;
            TD[x]+=FD[u]*complex<double>(cos(angle),sin(angle));
        }

    }


    }
```

③对图像进行傅里叶变换和反变换的函数代码

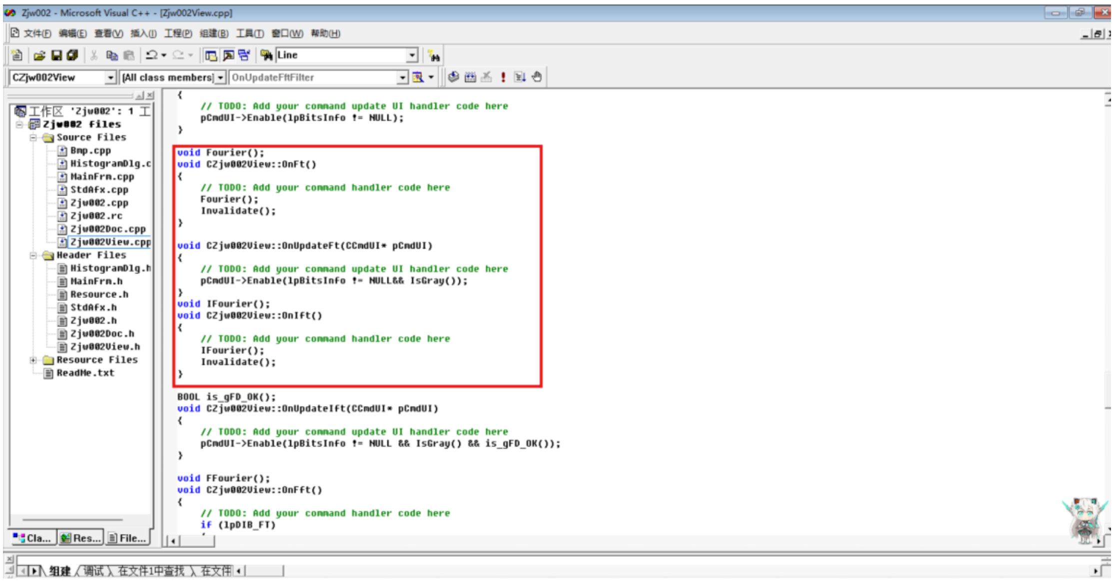
```
void Fourier(){
    int w = lpBitsInfo->bmiHeader.biWidth;
    int h = lpBitsInfo->bmiHeader.biHeight;
    int LineBytes = (w * lpBitsInfo->bmiHeader.biBitCount + 31) / 32 *
4;
```

```
    BYTE* lpBits =
(BYTE*)&lpBitsInfo->bmiColors[lpBitsInfo->bmiHeader.biClrUsed];

    complex<double >* TD = new complex<double>[w*h];
    complex<double >* FD = new complex<double>[w*h];

    int i,j;
    BYTE* pixel;
    for(i=0;i<h;i++){
        for(j=0;j<w;j++){
            pixel = lpBits+LineBytes*(h-1-i)+j;
            TD[i*w+j] = complex<double>(*pixel * pow(-1,i+j) , 0);
        }
    }
    for(i = 0;i<h;i++){
        FT(&TD[w*i],&FD[w*i],w);
    }
    for(i=0;i<h;i++){
        for(j=0;j<w;j++){
            TD[j*h+i] = FD[i*w+j];
        }
    }
    for(i=0;i<w;i++){
        FT(&TD[h*i],&FD[h*i],h);
    }

    DWORD Size = 40 + 1024+LineBytes*h;

    lpDIB_FT=(BITMAPINFO*)malloc(Size);
    memcpy(lpDIB_FT,lpBitsInfo,Size);

    lpBits = (BYTE*)&lpDIB_FT->bmiColors[256];
    double temp;
    for(i=0;i<h;i++){
        for(j=0;j<w;j++){
            pixel =lpBits+LineBytes*(h-1-i)+j;
            temp=
sqrt(FD[j*h+i].real()*FD[j*h+i].real()+FD[j*h+i].imag()*FD[j*h+i].ima
g())*1000;
            if(temp>255)
                temp = 255;
            *pixel  = (BYTE)(temp);
        }
    }
```

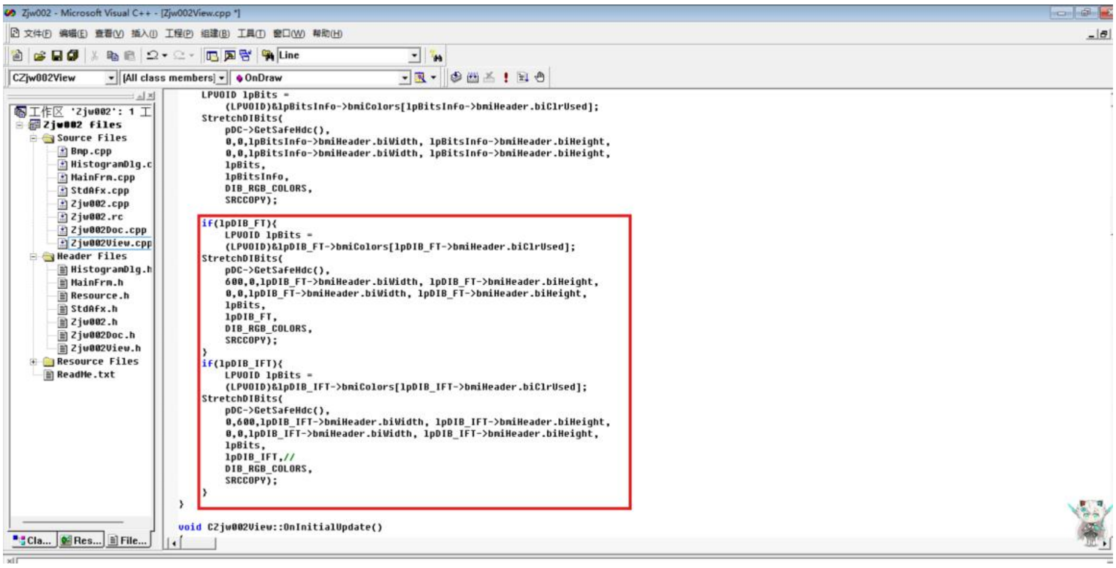
```
    delete TD;
    //delete FD;
    gFD = FD;
}

void IFourier(){
    int w = lpBitsInfo->bmiHeader.biWidth;
    int h = lpBitsInfo->bmiHeader.biHeight;
    int LineBytes = (w * lpBitsInfo->bmiHeader.biBitCount + 31) / 32 *
4;
    BYTE* lpBits =
(BYTE*)&lpBitsInfo->bmiColors[lpBitsInfo->bmiHeader.biClrUsed];
    complex<double >* TD = new complex<double>[w*h];
    int i,j;



    for(i = 0;i<w;i++){
        IFT(&gFD[h*i],&TD[h*i],h);
    }
    for(i=0;i<h;i++){
        for(j=0;j<w;j++){
            gFD[j*h+i] = TD[i*w+j];
        }
    }
    for(i=0;i<h;i++){
        IFT(&gFD[w*i],&TD[w*i],w);
    }




    DWORD Size = 40 + 1024+LineBytes*h;

    lpDIB_IFT=(BITMAPINFO*)malloc(Size);
    memcpy(lpDIB_IFT,lpBitsInfo,Size);

    lpBits = (BYTE*)&lpDIB_IFT->bmiColors[256];

    BYTE *pixel;

    for(i=0;i<h;i++){
        for(j=0;j<w;j++){
            pixel =lpBits+LineBytes*(h-1-i)+j;
```

```
            *pixel  = (BYTE)(TD[w*i+j].real()/pow(-1,i+j));
        }
    }
    delete TD;
    delete gFD;
    gFD=NULL;
}
```

b.在视图类中调用之



c.在视图类的 onDraw 函数添加绘制代码，使其可以正确地绘制变换后的图像



2.FFT&IFFT

a. 在 bmp 文件中添加两个函数分别实现快速傅里叶变换和反变换计算方法，

然后再添加两个函数对图像进行快速傅里叶变换和反变换。代码如下

①快速傅里叶变换和反变换的计算公式代码实现

```
BOOL is_gFD_OK(){
    return (gFD != NULL);
}


//快速傅里叶变换
void FFT(complex<double> * TD, complex<double> * FD, int r)
{
    // 计算付立叶变换点数
    LONG count = 1 << r;
    // 计算加权系数
    int i;
    double angle;
    complex<double>* W = new complex<double>[count / 2];
    for(i = 0; i < count / 2; i++)
    {
        angle = -i * PI * 2 / count;
        W[i] = complex<double> (cos(angle), sin(angle));
    }
    // 将时域点写入X1
    complex<double>* X1 = new complex<double>[count];
    memcpy(X1, TD, sizeof(complex<double>) * count);

    // 采用蝶形算法进行快速付立叶变换，输出为频域值X2
    complex<double>* X2 = new complex<double>[count];

    int k,j,p,size;
    complex<double>* temp;
    for (k = 0; k < r; k++)
    {
        for (j = 0; j < 1 << k; j++)
        {
            size = 1 << (r-k);
            for (i = 0; i < size/2; i++)
            {
                p = j * size;
                X2[i + p] = X1[i + p] + X1[i + p + size/2];
                X2[i + p + size/2] = (X1[i + p] - X1[i + p + size/2]) * W[i
* (1<<k)];
            }
        }
```

```
        temp  = X1;
        X1 = X2;
        X2 = temp;
    }

    // 重新排序（码位倒序排列）
    for (j = 0; j < count; j++)
    {
        p = 0;
        for (i = 0; i < r; i++)
        {
            if (j & (1<<i))
            {
                p += 1<<(r-i-1);
            }
        }
        FD[j]=X1[p];
        FD[j] /= count;
    }

    // 释放内存
    delete W;
    delete X1;
    delete X2;
    }
/快速傅里叶反变换
//IFFT反变换
void IFFT(complex<double> * FD, complex<double> * TD, int r)
{
    // 付立叶变换点数
    LONG   count;
    // 计算付立叶变换点数
    count = 1 << r;

    // 分配运算所需存储器
    complex<double> * X = new complex<double>[count];
    // 将频域点写入X
    memcpy(X, FD, sizeof(complex<double>) * count);

    // 求共轭
    for(int i = 0; i < count; i++)
        X[i] = complex<double> (X[i].real(), -X[i].imag());

    // 调用快速付立叶变换
```

```
    FFT(X, TD, r);

    // 求时域点的共轭
    for(i = 0; i < count; i++)
        TD[i] = complex<double> (TD[i].real() * count, -TD[i].imag() *
count);

    // 释放内存
    delete X;
    }
```

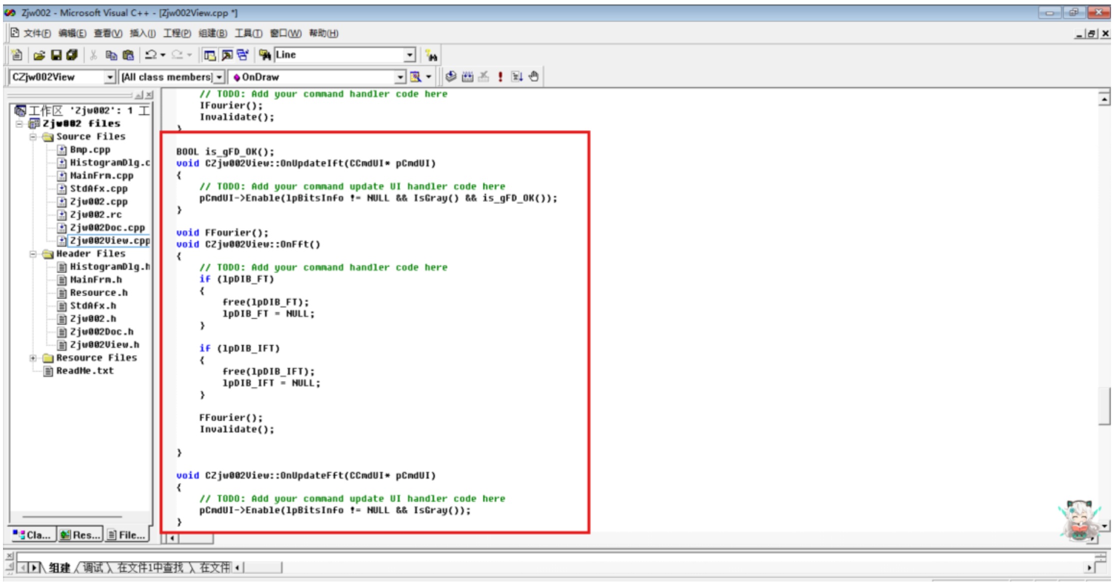
②对图像进行傅里叶变换和反变换的函数代码

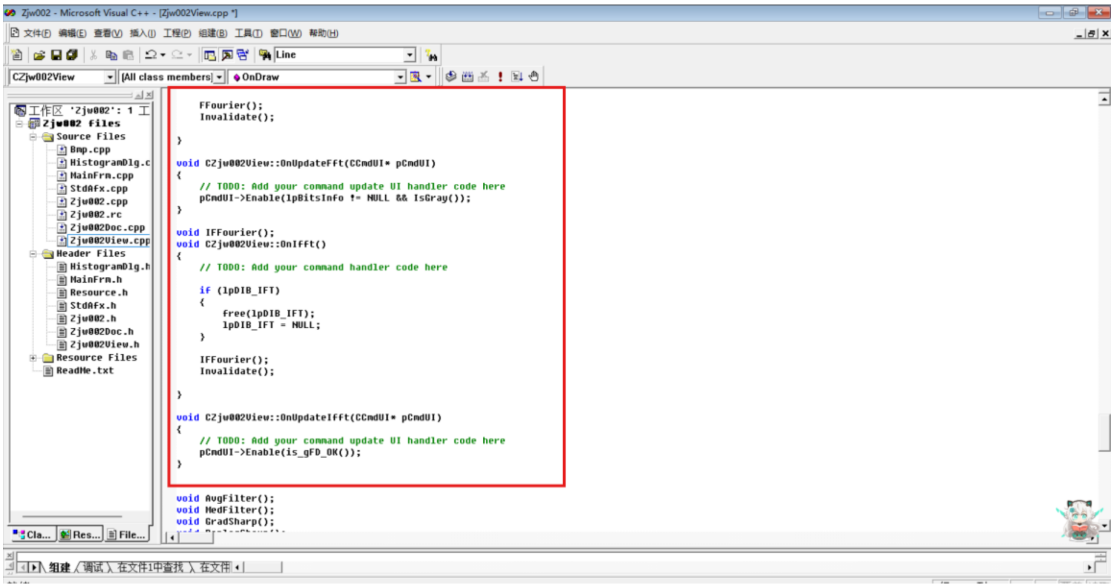
```
void FFourier()
{
    //图像的宽度和高度
    int width = lpBitsInfo->bmiHeader.biWidth;
    int height = lpBitsInfo->bmiHeader.biHeight;
    int LineBytes = (width * lpBitsInfo->bmiHeader.biBitCount + 31)/32
* 4;
    //指向图像数据指针
    BYTE* lpBits =
(BYTE*)&lpBitsInfo->bmiColors[lpBitsInfo->bmiHeader.biClrUsed];

    // FFT宽度（必须为2的整数次方）
    int FFT_w = 1;
    // FFT宽度的幂数，即迭代次数
    int wp = 0;
    while(FFT_w * 2 <= width)
    {
        FFT_w *= 2;
        wp ++;
    }

    // FFT高度（必须为2的整数次方）
    int FFT_h = 1;
    // FFT高度的幂数，即迭代次数
    int hp = 0;
    while(FFT_h * 2 <= height)
    {
        FFT_h *= 2;
        hp ++;
    }

    // 分配内存
```

```
complex<double>* TD = new complex<double>[FFT_w * FFT_h];
complex<double>* FD = new complex<double>[FFT_w * FFT_h];

int i, j;
BYTE* pixel;

for(i = 0; i < FFT_h; i++)  // 行
{
    for(j = 0; j < FFT_w; j++)  // 列
    {
        // 指向DIB第i行，第j个象素的指针
        pixel = lpBits + LineBytes * (height - 1 - i) + j;

        // 给时域赋值
        TD[j + FFT_w * i] = complex<double>(*pixel* pow(-1,i+j), 0);
    }
}

for(i = 0; i < FFT_h; i++)
{
    // 对y方向进行快速付立叶变换
    FFT(&TD[FFT_w * i], &FD[FFT_w * i], wp);
}

// 保存中间变换结果
for(i = 0; i < FFT_h; i++)
{
    for(j = 0; j < FFT_w; j++)
    {
        TD[i + FFT_h * j] = FD[j + FFT_w * i];
    }
}

for(i = 0; i < FFT_w; i++)
{
    // 对x方向进行快速付立叶变换
    FFT(&TD[i * FFT_h], &FD[i * FFT_h], hp);
}

//生成频谱图像
//为频域图像分配内存
LONG size = 40 + 1024 + LineBytes * height;
lpDIB_FT = (LPBITMAPINFO) malloc(size);
if (NULL == lpDIB_FT)
```

```
        return;
    memcpy(lpDIB_FT, lpBitsInfo, size);

    //指向频域图像数据指针
    lpBits =
(BYTE*)&lpDIB_FT->bmiColors[lpDIB_FT->bmiHeader.biClrUsed];

    double temp;
    for(i = 0; i < FFT_h; i++) // 行
    {
        for(j = 0; j < FFT_w; j++) // 列
        {
            // 计算频谱幅度
            temp = sqrt(FD[j * FFT_h + i].real() * FD[j * FFT_h + i].real()
+
                        FD[j * FFT_h + i].imag() * FD[j * FFT_h + i].imag())
*2000;

            // 判断是否超过255
            if (temp > 255)
            {
                // 对于超过的，直接设置为255
                temp = 255;
            }

            pixel = lpBits + LineBytes * (height - 1 - i) + j;

            // 更新源图像
            *pixel = (BYTE)(temp);
        }
    }

    delete TD;
    gFD = FD;
    }
```

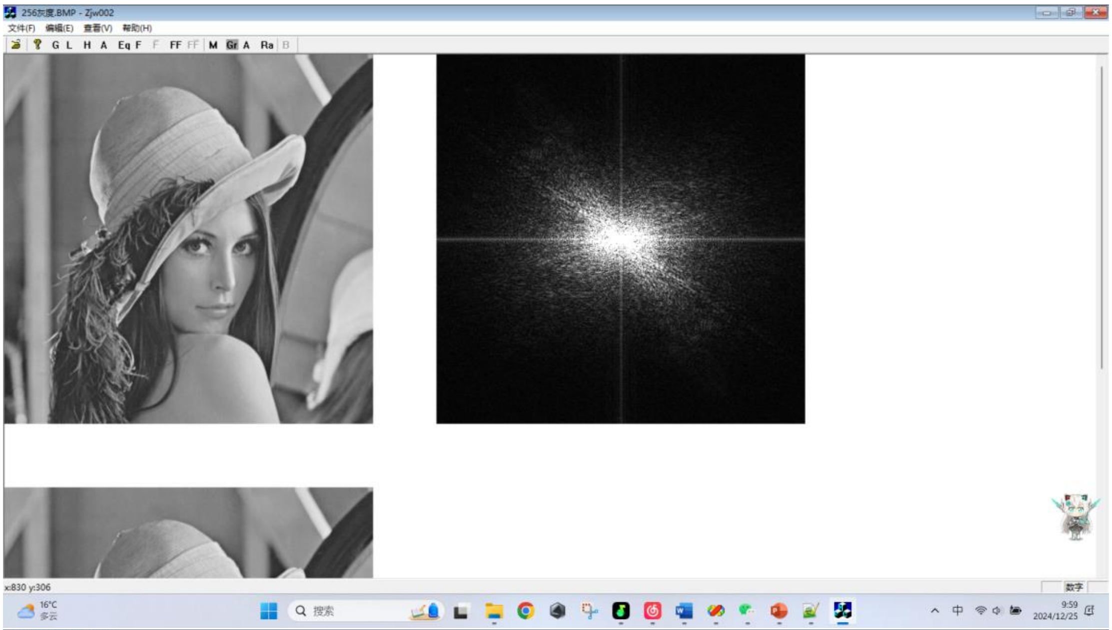
b.在视图类中调用之

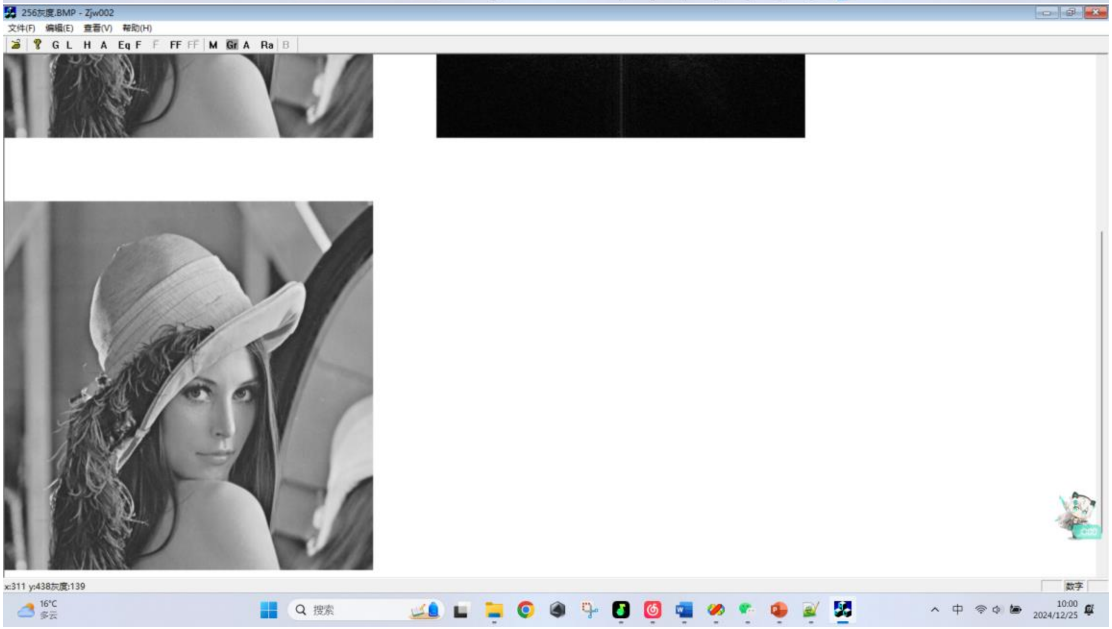




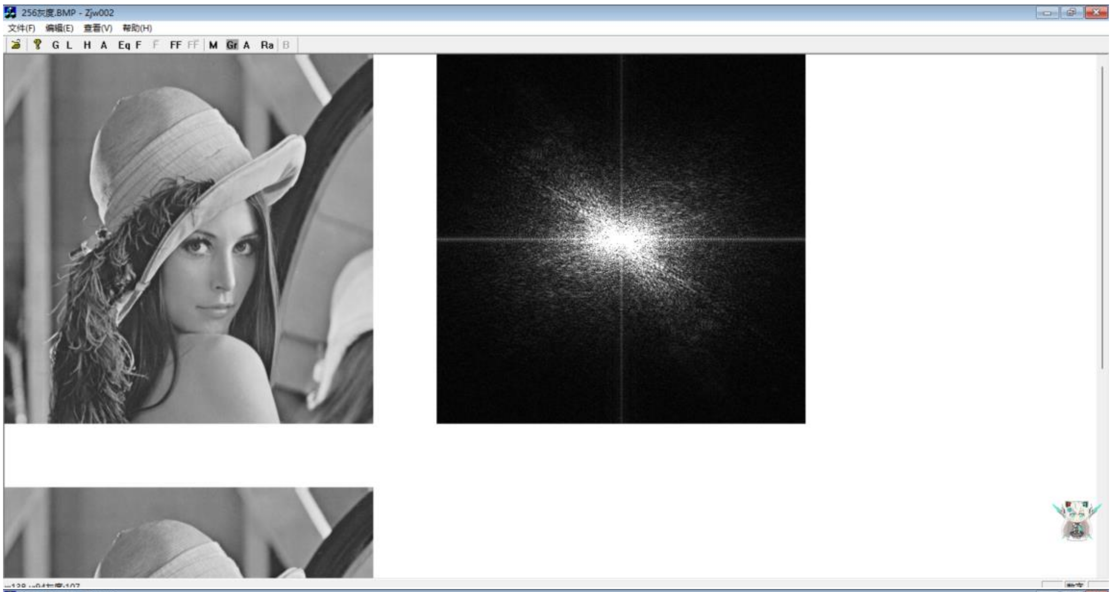
（五） 实验结果与分析

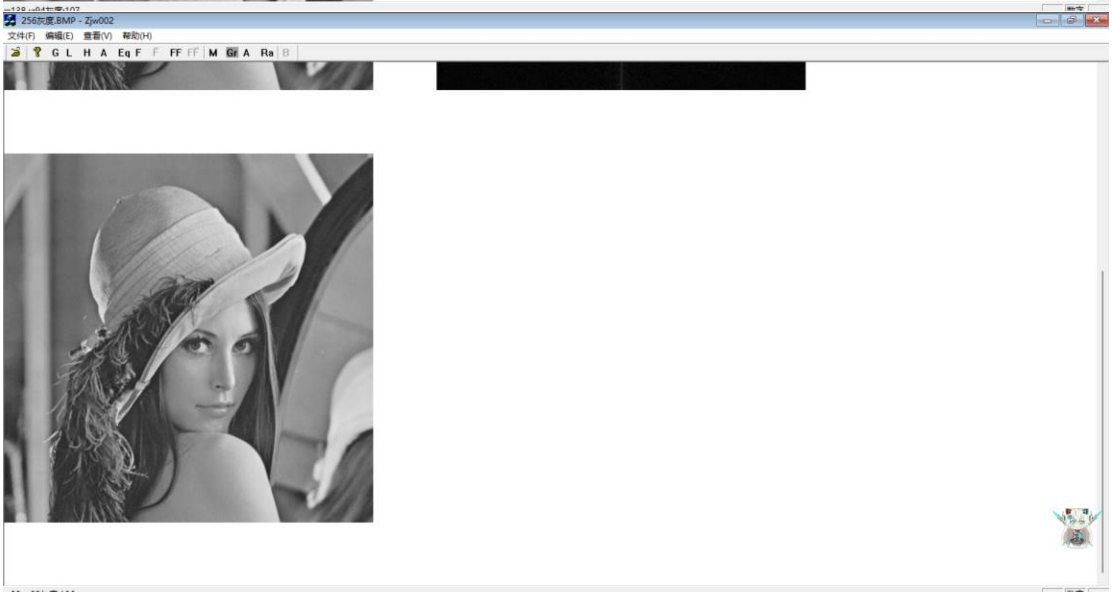
1.傅里叶变换和反变换


256灰度.BMP - Zjw002
文件(F) 编辑(E) 查看(V) 帮助(H)
x:830 y:306


256灰度.BMP - Zjw002
文件(F) 编辑(E) 查看(V) 帮助(H)
x:311 y:438灰度:139

2.快速傅里叶变换





3.实验结果分析

通过本次实验，我掌握了傅里叶变换和反变换的计算方法以及快速傅里叶变换和反变换的计算方法，并利用代码实现了以上算法，将其运用到实际的图像处理中，让我理解了尤其是快速傅里叶变换和反变换在工程上的作用。