```
#include "stdafx.h"
#include "funtions.h"


std::vector<Point>  mousePoints;
Point points;
/**********************************************************************************************************************
傅里叶变换demo（主函数）
**********************************************************************************************************************/

int dftDemo() {

cv::Mat srcMat = imread("../testImages\\rose.jpg", 0);
cv::Mat magMat;

if (srcMat.empty()) {
std::cout << "failed to read image!:" << std::endl;
return -1;
}

//把图像转换为可视的傅里叶变换图像
calcVisibalMag(srcMat, magMat);

imshow("Input Image", srcMat);    // Show the result
imshow("spectrum magnitude", magMat);
waitKey(0);

return 0;

}

/*****************************************
1.输入一张图片，计算其可视化的幅值谱
2.再幅值谱图上，通过鼠标选择需要去除的频率
3.去除被选择的信号，然后复原图像
*****************************************/
int removeFrequnce()
{
cv::Mat srcMat = imread("../testImages\\rose.jpg", 0);
cv::Mat magMat;
cv::Mat phMat;
cv::Mat maskMat;
double normVal;

if (srcMat.empty()) {
std::cout << "failed to read image!:" << std::endl;
return -1;
}

//输出可视化的mag，以及相位谱，以及归一化系数
calcVisbalDft(srcMat, magMat, phMat, normVal);

//在幅值谱上，通过鼠标选择，需要去掉的频率
selectPolygon(magMat, maskMat);

//逆变换


return 0;
}


/*************************************鼠标响应函数*****************************************/
void on_mouse(int EVENT, int x, int y, int flags, void* userdata)
{

Mat hh;
hh = *(Mat*)userdata;
Point p(x, y);
switch (EVENT)
{
case EVENT_LBUTTONDOWN:
{
points.x = x;
points.y = y;
mousePoints.push_back(points);
circle(hh, points, 4, cvScalar(255, 255, 255), -1);
imshow("mouseCallback", hh);
}
break;
}

}


int selectPolygon(cv::Mat srcMat, cv::Mat &dstMat)
{

vector<vector<Point>> contours;
cv::Mat selectMat;

cv::Mat m = cv::Mat::zeros(srcMat.size(), CV_32F);
```

```
m = 1;

if (!srcMat.empty()) {
srcMat.copyTo(selectMat);
srcMat.copyTo(dstMat);
}
else {
std::cout << "failed to read image!:" << std::endl;
return -1;
}

namedWindow("mouseCallback");
imshow("mouseCallback", selectMat);
setMouseCallback("mouseCallback", on_mouse, &selectMat);
waitKey(0);
destroyAllWindows();
//计算roi
contours.push_back(mousePoints);
if (contours[0].size() < 3) {
std::cout << "failed to read image!:" << std::endl;
return -1;
}

drawContours(m, contours, 0, Scalar(0), -1);

m.copyTo(dstMat);

return 0;
}


/***********输入一张图片，输出其傅里叶变换后的可视化的幅值谱*******************/
int calcVisibalMag(cv::Mat srcMat, cv::Mat & dstMat)
{

if (srcMat.empty()) {
std::cout << "failed to read image!:" << std::endl;
return -1;
}

Mat padMat;
//当图像的尺寸是2，3，5的整数倍时，离散傅里叶变换的计算速度最快。
//获得输入图像的最佳变换尺寸
int m = getOptimalDFTSize(srcMat.rows);
int n = getOptimalDFTSize(srcMat.cols);


//对新尺寸的图片进行边缘边缘填充
/************************************************
copyMakeBorder（） 函数模型：
copyMakeBorder(InputArray src, OutputArray dst,
                                int top, int bottom, int left, int right,
                                int borderType, const Scalar& value = Scalar() );
参数介绍：
. InputArray src：InputArray类型的src
. OutputArray dst：输出图像
. int top, int bottom, int left, int right：表示对边界每个方向添加的像素个数
. int borderType：表示扩充边界的类型，如BORDER_REPLICATE 就是对边界像素进行复制， BORDER_REFLECT 反射：对感兴趣的图像中的像素在两边进行复制

. const Scalar& value :边界的颜色值
************************************************/
copyMakeBorder(srcMat, padMat, 0, m - srcMat.rows, 0, n - srcMat.cols, BORDER_CONSTANT, Scalar::all(0));




//定义一个数组,存储频域转换成float类型的对象，再存储一个和它一样大小空间的对象来存储复数部分
Mat planes[] = { Mat_<float>(padMat), Mat::zeros(padMat.size(), CV_32F) };
Mat complexMat;

//将2个单通道的图像合成一幅多通道图像
/************************************************
merge（） 函数模型：
merge(const Mat* mv, size_t count, OutputArray dst);
参数介绍：
. const Mat* mv：Mat型数组
. size_t count：需合并数组个数
. OutputArray dst：输出矩阵
************************************************/
merge(planes, 2, complexMat);


//进行傅里叶变换,结果保存在原Mat里,傅里叶变换结果为复数.通道1存的是实部,通道二存的是虚部
/************************************************
dft（） 函数模型：
dft(InputArray src, OutputArray dst, int flags = 0, int nonzeroRows = 0);
参数介绍：
. InputArray src：输入图像，可以是实数或虚数
. OutputArray dst：输出图像，其大小和类型取决于第三个参数flags
. int flags：转换的标识符，有默认值0
. int nonzeroRows：当这个参数不为0，函数会假设只有输入数组（没有设置DFT_INVERSE）的第一行或第一个输出数组（设置了DFT_INVERSE）包含非零值。
************************************************/
dft(complexMat, complexMat);
```

```
//将双通道的图分离成量个单通道的图
//实部：planes[0] = Re(DFT(I),
//虚部：planes[1]=  Im(DFT(I)))
split(complexMat, planes);
//求相位，保存在planes[0]
magnitude(planes[0], planes[1], planes[0]);

//以下步骤均为了显示方便
Mat magMat = planes[0];
// log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))
magMat += Scalar::all(1);
//取对数
/***********************************************
log（） 函数模型：
log(InputArray src, OutputArray dst);
参数介绍：
. InputArray src：输入图像，可以是实数或虚数
. OutputArray dst：输出得到的对数值
***********************************************/
log(magMat, magMat);

//确保对称
magMat = magMat(Rect(0, 0, magMat.cols & -2, magMat.rows & -2));
int cx = magMat.cols / 2;
int cy = magMat.rows / 2;
//将图像移相
/*
0 | 1         3 | 2
-------  ===> -------
2 | 3         1 | 0
*/
Mat q0(magMat, Rect(0, 0, cx, cy));
Mat q1(magMat, Rect(cx, 0, cx, cy));
Mat q2(magMat, Rect(0, cy, cx, cy));
Mat q3(magMat, Rect(cx, cy, cx, cy));
Mat tmp;
q0.copyTo(tmp);
q3.copyTo(q0);
tmp.copyTo(q3);
q1.copyTo(tmp);
q2.copyTo(q1);
tmp.copyTo(q2);

//为了imshow可以显示，归一化到0和1之间
/***********************************************
normalize（） 函数模型：
normalize( InputArray src, InputOutputArray dst, double alpha = 1, double beta = 0,
                            int norm_type = NORM_L2, int dtype = -1, InputArray mask = noArray());
参数介绍：
InputArray src：输入数组
    InputOutputArray dst：输出数组，支持原地运算
double alpha：range normalization模式的最小值
double beta：range normalization模式的最大值，不用于norm normalization(范数归一化)模式。
normType：归一化的类型，可以有以下的取值：
  NORM_MINMAX:数组的数值被平移或缩放到一个指定的范围，线性归一化，一般较常用。
  NORM_INF:此类型的定义没有查到，根据OpenCV 1的对应项，可能是归一化数组的C-范数(绝对值的最大值)
  NORM_L1 : 归一化数组的L1-范数(绝对值的和)
  NORM_L2: 归一化数组的(欧几里德)L2-范数
dtype：dtype为负数时，输出数组的type与输入数组的type相同；否则，输出数组与输入数组只是通道数相同，而tpye=CV_MAT_DEPTH(dtype).
mask：操作掩膜，用于指示函数是否仅仅对指定的元素进行操作
***********************************************/
normalize(magMat, magMat, 0, 1, NORM_MINMAX);
magMat = magMat * 255;
magMat.copyTo(dstMat);

return 0;
}


//输入一张图片，输出其傅里叶变换后的可视化的幅值谱
//同时输出相位谱，和还原归一化时的系数，即最大值
int calcVisbalDft(cv::Mat srcMat, cv::Mat & magMat, cv::Mat & ph, double & normVal)
{
cv::Mat dst;
cv::Mat src = imread("../testImages\\rose.jpg", 0);

int m = getOptimalDFTSize(src.rows); //2,3,5的倍数有更高效率的傅里叶变换
int n = getOptimalDFTSize(src.cols);
Mat padded;
//把灰度图像放在左上角,在右边和下边扩展图像,扩展部分填充为0;
copyMakeBorder(src, padded, 0, m - src.rows, 0, n - src.cols, BORDER_CONSTANT, Scalar::all(0));
//planes[0]为dft变换的实部，planes[1]为虚部，ph为相位， plane_true=mag为幅值
Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
Mat planes_true = Mat_<float>(padded);

//保存相位（Mat_代表确定了数据类型，访问元素时不需要再指定元素类型）
ph = Mat_<float>(padded);

Mat complexImg;
//多通道complexImg既有实部又有虚部
```

```
merge(planes, 2, complexImg);
//对上边合成的mat进行傅里叶变换,***支持原地操作***,傅里叶变换结果为复数.通道1存的是实部,通道二存的是虚部
dft(complexImg, complexImg);
//把变换后的结果分割到两个mat,一个实部,一个虚部,方便后续操作
split(complexImg, planes);

//---------------此部分目的为更好地显示幅值---后续恢复原图时反着再处理一遍-------------------------
magnitude(planes[0], planes[1], planes_true);//幅度谱mag
phase(planes[0], planes[1], ph);//相位谱ph
Mat A = planes[0];
Mat B = planes[1];
Mat mag = planes_true;

mag += Scalar::all(1);//对幅值加1
  //计算出的幅值一般很大，达到10^4,通常没有办法在图像中显示出来，需要对其进行log求解。
log(mag, mag);

//取矩阵中的最大值，便于后续还原时去归一化
minMaxLoc(mag, 0, &normVal, 0, 0);

//修剪频谱,如果图像的行或者列是奇数的话,那其频谱是不对称的,因此要修剪
mag = mag(Rect(0, 0, mag.cols & -2, mag.rows & -2));
ph = ph(Rect(0, 0, mag.cols & -2, mag.rows & -2));
Mat _magI = mag.clone();
//将幅度归一化到可显示范围。
normalize(_magI, _magI, 0, 1, CV_MINMAX);
//imshow("before rearrange", _magI);

//显示规则频谱图
int cx = mag.cols / 2;
int cy = mag.rows / 2;

//这里是以中心为标准，把mag图像分成四部分
Mat tmp;
Mat q0(mag, Rect(0, 0, cx, cy));
Mat q1(mag, Rect(cx, 0, cx, cy));
Mat q2(mag, Rect(0, cy, cx, cy));
Mat q3(mag, Rect(cx, cy, cx, cy));
q0.copyTo(tmp);
q3.copyTo(q0);
tmp.copyTo(q3);
q1.copyTo(tmp);
q2.copyTo(q1);
tmp.copyTo(q2);

normalize(mag, mag, 0, 1, CV_MINMAX);
mag = mag * 255;

return 0;
}

int calcDft2Image(cv::Mat magMat, cv::Mat ph, double normVal, cv::Mat & dstMat)
{
Mat mag=magMat.clone();
Mat proceMag;
//planes[0]为dft变换的实部，planes[1]为虚部，ph为相位， plane_true=mag为幅值
Mat planes[] = { Mat_<float>(mag), Mat::zeros(mag.size(), CV_32F) };
Mat planes_true = Mat_<float>(mag);

Mat complexImg;
//多通道complexImg既有实部又有虚部

mag = magMat/ 255;

proceMag = mag * 255;

int cx = mag.cols / 2;
int cy = mag.rows / 2;
//前述步骤反着来一遍，目的是为了逆变换回原图
Mat q00(mag, Rect(0, 0, cx, cy));
Mat q10(mag, Rect(cx, 0, cx, cy));
Mat q20(mag, Rect(0, cy, cx, cy));
Mat q30(mag, Rect(cx, cy, cx, cy));

Mat tmp;
//交换象限
q00.copyTo(tmp);
q30.copyTo(q00);
tmp.copyTo(q30);
q10.copyTo(tmp);
q20.copyTo(q10);
tmp.copyTo(q20);

mag = mag * normVal;//将归一化的矩阵还原
exp(mag, mag); //对应于前述去对数
mag = mag - Scalar::all(1);//对应前述+1
polarToCart(mag, ph, planes[0], planes[1]);//由幅度谱mag和相位谱ph恢复实部planes[0]和虚部planes[1]
merge(planes, 2, complexImg);//将实部虚部合并


//-----------------------傅里叶的逆变换-----------------------------------
Mat ifft(Size(mag.cols, mag.rows), CV_8UC1);
//傅里叶逆变换
```

```
idft(complexImg, ifft, DFT_REAL_OUTPUT);
normalize(ifft, ifft, 0, 1, CV_MINMAX);

Rect rect(0, 0, mag.cols, mag.rows);
dstMat = ifft(rect);
dstMat = dstMat * 255;

/*cv::Mat dspMat;
dst.convertTo(dspMat, CV_8UC1);*/


return 0;
}



int mouseROI()
{
cv::Mat srcMat = imread("../testImages\\rose.jpg");
cv::Mat dstMat;

selectPolygon(srcMat, dstMat);

imshow("srcMat", srcMat);
imshow("select Area", dstMat);
waitKey(0);

return 0;
}



int ifftDemo()
{
cv::Mat dst;

cv::Mat src = imread("../testImages\\rose.jpg", 0);

int m = getOptimalDFTSize(src.rows); //2,3,5的倍数有更高效率的傅里叶变换
int n = getOptimalDFTSize(src.cols);
Mat padded;
//把灰度图像放在左上角,在右边和下边扩展图像,扩展部分填充为0;
copyMakeBorder(src, padded, 0, m - src.rows, 0, n - src.cols, BORDER_CONSTANT, Scalar::all(0));
//planes[0]为dft变换的实部，planes[1]为虚部，ph为相位， plane_true=mag为幅值
Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
Mat planes_true = Mat_<float>(padded);
Mat ph = Mat_<float>(padded);
Mat complexImg;
//多通道complexImg既有实部又有虚部
merge(planes, 2, complexImg);
//对上边合成的mat进行傅里叶变换,***支持原地操作***,傅里叶变换结果为复数.通道1存的是实部,通道二存的是虚部
dft(complexImg, complexImg);
//把变换后的结果分割到两个mat,一个实部,一个虚部,方便后续操作
split(complexImg, planes);

//---------------此部分目的为更好地显示幅值---后续恢复原图时反着再处理一遍-------------------------
magnitude(planes[0], planes[1], planes_true);//幅度谱mag
phase(planes[0], planes[1], ph);//相位谱ph
Mat A = planes[0];
Mat B = planes[1];
Mat mag = planes_true;

mag += Scalar::all(1);//对幅值加1
//计算出的幅值一般很大，达到10^4,通常没有办法在图像中显示出来，需要对其进行log求解。
log(mag, mag);

//取矩阵中的最大值，便于后续还原时去归一化
double maxVal;
minMaxLoc(mag, 0, &maxVal, 0, 0);

//修剪频谱,如果图像的行或者列是奇数的话,那其频谱是不对称的,因此要修剪
mag = mag(Rect(0, 0, mag.cols & -2, mag.rows & -2));
ph = ph(Rect(0, 0, mag.cols & -2, mag.rows & -2));
Mat _magI = mag.clone();
//将幅度归一化到可显示范围。
normalize(_magI, _magI, 0, 1, CV_MINMAX);
//imshow("before rearrange", _magI);

//显示规则频谱图
int cx = mag.cols / 2;
int cy = mag.rows / 2;

//这里是以中心为标准，把mag图像分成四部分
Mat tmp;
Mat q0(mag, Rect(0, 0, cx, cy));
Mat q1(mag, Rect(cx, 0, cx, cy));
Mat q2(mag, Rect(0, cy, cx, cy));
Mat q3(mag, Rect(cx, cy, cx, cy));
q0.copyTo(tmp);
q3.copyTo(q0);
tmp.copyTo(q3);
q1.copyTo(tmp);
q2.copyTo(q1);
```

```
tmp.copyTo(q2);

normalize(mag, mag, 0, 1, CV_MINMAX);
//imshow("原图灰度图", src);
//imshow("频谱幅度", mag);
mag = mag * 255;
imwrite("原频谱.jpg", mag);
/*-------------------------------------------------*/

mag = mag / 255;
cv::Mat mask;
Mat proceMag;

selectPolygon(mag,mask);

mag= mag.mul(mask);

proceMag = mag * 255;
imwrite("处理后频谱.jpg", proceMag);

//前述步骤反着来一遍，目的是为了逆变换回原图
Mat q00(mag, Rect(0, 0, cx, cy));
Mat q10(mag, Rect(cx, 0, cx, cy));
Mat q20(mag, Rect(0, cy, cx, cy));
Mat q30(mag, Rect(cx, cy, cx, cy));

//交换象限
q00.copyTo(tmp);
q30.copyTo(q00);
tmp.copyTo(q30);
q10.copyTo(tmp);
q20.copyTo(q10);
tmp.copyTo(q20);

mag = mag * maxVal;//将归一化的矩阵还原
exp(mag, mag);//对应于前述去对数
mag = mag - Scalar::all(1);//对应前述+1
polarToCart(mag, ph, planes[0], planes[1]);//由幅度谱mag和相位谱ph恢复实部planes[0]和虚部planes[1]
merge(planes, 2, complexImg);//将实部虚部合并


//----------------------傅里叶的逆变换-----------------------------------
Mat ifft(Size(src.cols, src.rows), CV_8UC1);
//傅里叶逆变换
idft(complexImg, ifft, DFT_REAL_OUTPUT);
normalize(ifft, ifft, 0, 1, CV_MINMAX);

Rect rect(0, 0, src.cols, src.rows);
dst = ifft(rect);
dst = dst * 255;

cv::Mat dspMat;
dst.convertTo(dspMat, CV_8UC1);
imshow("dst", dspMat);
imshow("src", src);
waitKey(0);

return 0;

}
```