谭树杰 11849060

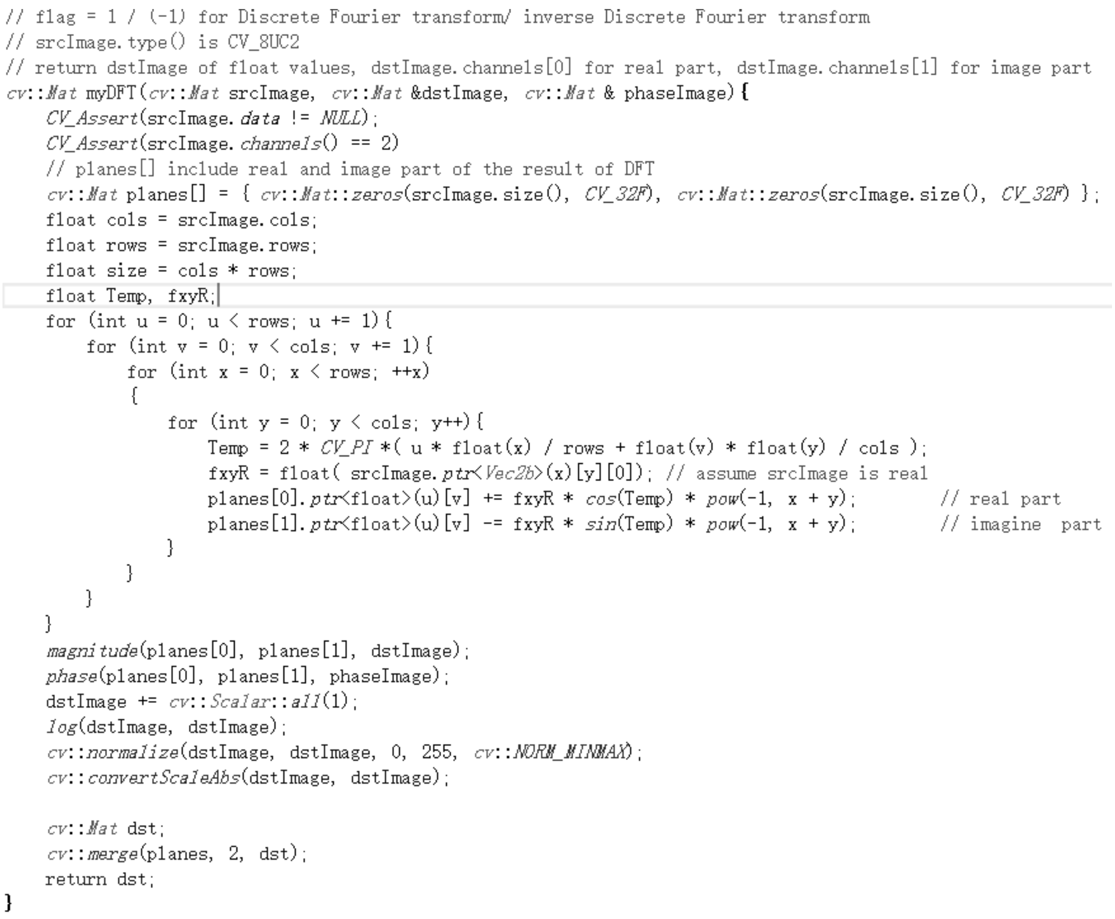
Lab4 Report

The main() function of this project is shown below

```
int main()
{
    // 1. Implement 2D DFT for all provided images and analyze their
    // transformed images in frequency domain
    // I transform the pgm to jpg by GIMP first
    cv::Mat srcImage = imread("..\\images\\bridge.jpg"); // bridge
    if (srcImage.empty())
    return-1;
    imshow("srcImage", srcImage);
    cv::Mat resultImage = DFT(srcImage);
    imshow("resultImage", resultImage);
    imwrite("..\\images\\bridge_mag.jpg", resultImage);
    srcImage = imread("..\\images\\lena.jpg");           // lena
    resultImage = DFT(srcImage);
    imshow("circles_mag.jpg", resultImage);
    imwrite("..\\images\\circles_mag.jpg", resultImage);
    srcImage = imread("..\\images\\circles.jpg");        // circles
    resultImage = DFT(srcImage);
    imshow("circles_mag.jpg", resultImage);
    imwrite("..\\images\\circles_mag.jpg", resultImage);
    srcImage = imread("..\\images\\crosses.jpg");        // crosses
    resultImage = DFT(srcImage);
    imshow("crosses_mag.jpg", resultImage);
    imwrite("..\\images\\crosses_mag.jpg", resultImage);
    srcImage = imread("..\\images\\goldhill.jpg");       // goldhill
    resultImage = DFT(srcImage);
    imshow("goldhill_mag.jpg", resultImage);
    imwrite("..\\images\\goldhill_mag.jpg", resultImage);
    srcImage = imread("..\\images\\horiz.jpg");          // horiz
    resultImage = DFT(srcImage);
    imshow("horiz_mag.jpg", resultImage);
    imwrite("..\\images\\horiz_mag.jpg", resultImage);
    srcImage = imread("..\\images\\montage.jpg");        // montage
    resultImage = DFT(srcImage);
    imshow("montage_mag.jpg", resultImage);
    imwrite("..\\images\\montage_mag.jpg", resultImage);
    // 2. Obtain the magnitude image and phase images of the
    // Lena.pgm and analyze the resulted images.
    cv::Mat srcImage = imread("..\\images\\lena.jpg");
    if (srcImage.empty())
        return-1;
    cv::Mat phaseImage;
    imshow("srcImage", srcImage);
    cv::Mat magnitudeImage = DFT(srcImage, phaseImage);    // magnitudeImage is the
magnitude image
    imshow("lena_mag", magnitudeImage);
    imshow("lena_phase", phaseImage);
    imwrite("..\\images\\lena_mag.jpg", magnitudeImage);
    imwrite("..\\images\\lena_phase.jpg", phaseImage);

    // 3. Reconstruct lena.pgm using the magnitude and phase images in
    // frequency domain respectively and analyze the results
    cv::Mat orginImage;
    myIDFT(magnitudeImage, phaseImage, orginImage);
    normalize(orginImage, orginImage, 0, 1, CV_MINMAX);
    imshow("orignImage", orginImage);
    imwrite("lena_origin.jpg", orginImage);
    /*
    cv::waitKey(0);
    return 0;
}
```

1. Implement 2D DFT for all provided images and analyze their transformed images in frequency domain :

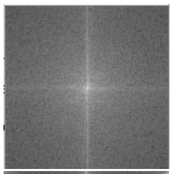
My implementation code is shown below:

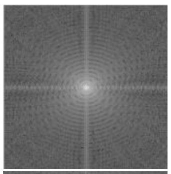


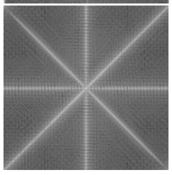
```
// flag = 1 / (-1) for Discrete Fourier transform/ inverse Discrete Fourier transform
// srcImage.type() is CV_8UC2
// return dstImage of float values, dstImage.channels[0] for real part, dstImage.channels[1] for image part
cv::Mat myDFT(cv::Mat srcImage, cv::Mat &dstImage, cv::Mat & phaseImage){
    CV_Assert(srcImage.data != NULL);
    CV_Assert(srcImage.channels() == 2)
    // planes[] include real and image part of the result of DFT
    cv::Mat planes[] = { cv::Mat::zeros(srcImage.size(), CV_32F), cv::Mat::zeros(srcImage.size(), CV_32F) };
    float cols = srcImage.cols;
    float rows = srcImage.rows;
    float size = cols * rows;
    float Temp, fxyR;
    for (int u = 0; u < rows; u += 1){
        for (int v = 0; v < cols; v += 1){
            for (int x = 0; x < rows; ++x)
            {
                for (int y = 0; y < cols; y++){
                    Temp = 2 * CV_PI *( u * float(x) / rows + float(v) * float(y) / cols );
                    fxyR = float( srcImage.ptr<Vec2b>(x)[y][0]); // assume srcImage is real
                    planes[0].ptr<float>(u)[v] += fxyR * cos(Temp) * pow(-1, x + y);        // real part
                    planes[1].ptr<float>(u)[v] -= fxyR * sin(Temp) * pow(-1, x + y);        // imagine  part
                }
            }
        }
    }
    magnitude(planes[0], planes[1], dstImage);
    phase(planes[0], planes[1], phaseImage);
    dstImage += cv::Scalar::all(1);
    log(dstImage, dstImage);
    cv::normalize(dstImage, dstImage, 0, 255, cv::NORM_MINMAX);
    cv::convertScaleAbs(dstImage, dstImage);

    cv::Mat dst;
    cv::merge(planes, 2, dst);
    return dst;
}
```

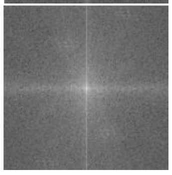
But there’s a bug in it, so the result correct enough. Therefor I use the code in 《OpenCV 图像处理编程实例》 to analysis DFT.

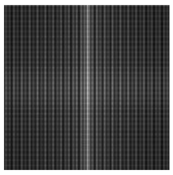
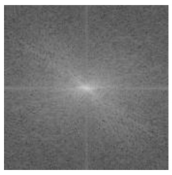
bridge.jpg, circles.jpg, crosses.jpg, goldhill.jpg, horiz.jpg, lena.jpg, montage.jpg and their magnitude images after dft transformation are shown below, respectively

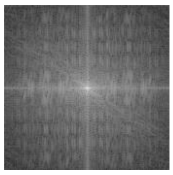










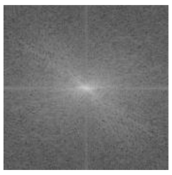
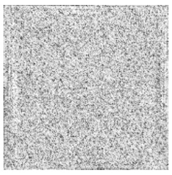
Analysis: We could only see bright lines in crosses.jpg and horiz.jpg, because these two images have a black background and several bright lines in it. The intensity changes dramatically in the areas of lines. In contrast, other magnitude images just have a bright dot in the center and with gray background. Because these images have a slower intensity transition.

2. **Obtain the magnitude image and phase images of the Lena.pgm and analyze the resulted images.**

   The implementation code is just as above.

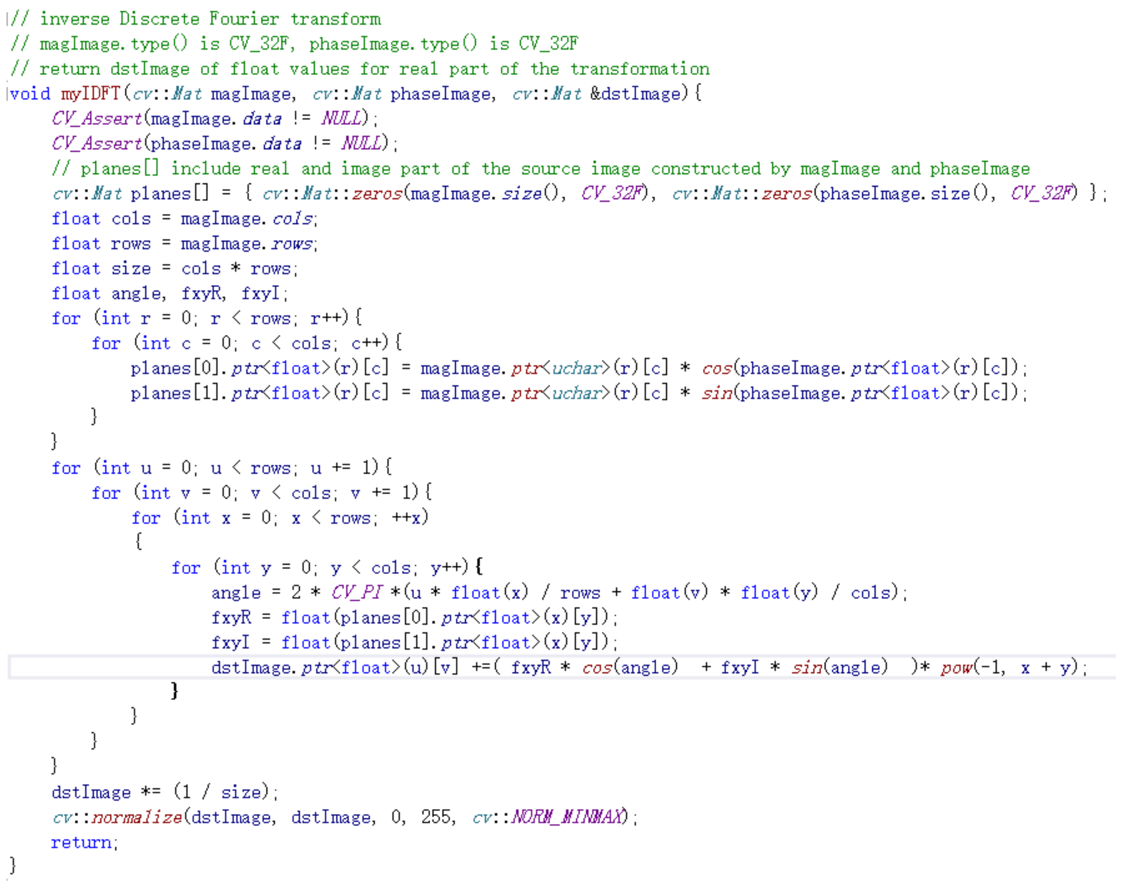
   The lena.jpg and its magnitude and phase image are shown below

   Analysis: The lena.jpg's intensity only changes dramatically in her hat's edge and her arm. So the magnitude image only has several bright line across the image. And it’s hard to get some intuitive information from the phase image.

3. Reconstruct lena.pgm using the magnitude and phase images in frequency domain respectively and analyze the results.

   The implementation code is shown below. But it does not work well enough too.



```cpp
// inverse Discrete Fourier transform
// magImage.type() is CV_32F, phaseImage.type() is CV_32F
// return dstImage of float values for real part of the transformation
void myIDFT(cv::Mat magImage, cv::Mat phaseImage, cv::Mat &dstImage){
    CV_Assert(magImage.data != NULL);
    CV_Assert(phaseImage.data != NULL);
    // planes[] include real and image part of the source image constructed by magImage and phaseImage
    cv::Mat planes[] = { cv::Mat::zeros(magImage.size(), CV_32F), cv::Mat::zeros(phaseImage.size(), CV_32F) };
    float cols = magImage.cols;
    float rows = magImage.rows;
    float size = cols * rows;
    float angle, fxyR, fxyI;
    for (int r = 0; r < rows; r++){
        for (int c = 0; c < cols; c++){
            planes[0].ptr<float>(r)[c] = magImage.ptr<uchar>(r)[c] * cos(phaseImage.ptr<float>(r)[c]);
            planes[1].ptr<float>(r)[c] = magImage.ptr<uchar>(r)[c] * sin(phaseImage.ptr<float>(r)[c]);
        }
    }
    for (int u = 0; u < rows; u += 1){
        for (int v = 0; v < cols; v += 1){
            for (int x = 0; x < rows; ++x)
            {
                for (int y = 0; y < cols; y++){
                    angle = 2 * CV_PI *(u * float(x) / rows + float(v) * float(y) / cols);
                    fxyR = float(planes[0].ptr<float>(x)[y]);
                    fxyI = float(planes[1].ptr<float>(x)[y]);
                    dstImage.ptr<float>(u)[v] +=( fxyR * cos(angle)  + fxyI * sin(angle)  )* pow(-1, x + y);
                }
            }
        }
    }
    dstImage *= (1 / size);
    cv::normalize(dstImage, dstImage, 0, 255, cv::NORM_MINMAX);
    return;
}
```

Analysis: The image doesn't as well as original image. Maybe due to the DFT transformation lost some information of the image.