Assignment7

Part 1 code

详见 FourierTransform.cpp。

Part 2 explanation

(1)傅里叶变换

对应的公式：

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

这里是代码中的实现：

```
for (int row = 0; row < height; row++)
{
    for (int col = 0; col < width; col++)
    {
        real = real + gray_value[row][col] * cos(2 * PI*(u*row / (double)height + v*col / (double)width));
        imag = imag - gray_value[row][col] * sin(2 * PI*(u*row / (double)height + v*col / (double)width));
    }
}
real = real / (double)(height * width);
imag = imag / (double)(height * width);
return MyComplex(real, imag);
```

(2)逆傅里叶变换

对应的公式：

$$f(x,y)=\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u,v)e^{-j2\pi(ux/M+vy/N)} \qquad \begin{matrix} x=0,1,\cdots M-1 \\ y=0,1,\cdots N-1 \end{matrix}$$

代码中的实现：

```
for (int row = 0; row < height; row++)
{
    for (int col = 0; col < width; col++)
    {
        real = real + gray_value[row][col] * cos(2 * PI*(u*row / (double)height + v*col / (double)width));
        imag = imag - gray_value[row][col] * sin(2 * PI*(u*row / (double)height + v*col / (double)width));
    }
}
return MyComplex(real, imag);
```

(3)载入原图

在载入原图时，对灰度做了变换，使得最后的频谱图的原点在图片的中心。

```
//get gray value of image
vector<vector<double>> gray_value;
for (int row = 0; row < img->height; row++)
{
    uchar* ptr = (uchar*)img->imageData + row*img->widthStep;
    vector<double> tempv;
    for (int col = 0; col < img->width; col++)
    {
        tempv.push_back(ptr[col] * pow(-1, row + col));//原点移到图像中心
    }
    gray_value.push_back(tempv);
}
```

(4)计算频谱图：

对应的公式：（这里 R(u,v)是实部，I(u,v)是虚部）

频谱/幅度谱/模 $$|F(u,v)| = \sqrt{R^2(u,v) + I^2(u,v)}$$

代码中的实现：

```
//最大频率和最小频率
int max_frequency = -1;
int min_frequency = -1;

//Spectrum
IplImage* spectrum = cvCloneImage(img);
for (int row = 0; row < spectrum->height; row++)
{
    uchar* ptr = (uchar*)spectrum->imageData + row*spectrum->widthStep;
    for (int col = 0; col < spectrum->width; col++)
    {
        int temp_frequency = sqrt(pow(frequency_value[row][col].real, 2) + pow(frequency_value[row][col].imag, 2));
        if (max_frequency == -1 || temp_frequency > max_frequency)max_frequency = temp_frequency;
        if (min_frequency == -1 || temp_frequency < min_frequency)min_frequency = temp_frequency;
        ptr[col] = temp_frequency;
    }
}
cvSaveImage("Spectrum.png", spectrum);
cvReleaseImage(&spectrum);
```

其中 max_frequency 和 min_frequency 是最大和最小频率。

(5)计算相位谱图

对应的公式：

相位谱 $$\phi(u,v) = \arctan\frac{I(u,v)}{R(u,v)}$$

代码中的实现：

```
//Phase Spectrum
IplImage* phaseSpectrum = cvCloneImage(img);
for (int row = 0; row < phaseSpectrum->height; row++)
{
    uchar* ptr = (uchar*)phaseSpectrum->imageData + row*phaseSpectrum->widthStep;
    for (int col = 0; col < phaseSpectrum->width; col++)
    {
        ptr[col] = atan(frequency_value[row][col].imag / frequency_value[row][col].real);
    }
}
cvSaveImage("PhaseSpectrum.png", phaseSpectrum);
cvReleaseImage(&phaseSpectrum);
```

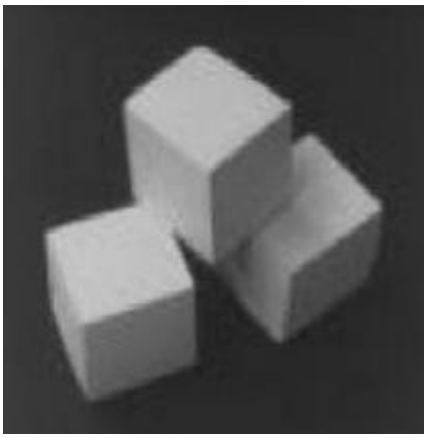
(6)滤波

输入保留的频率域，之后进行滤波：

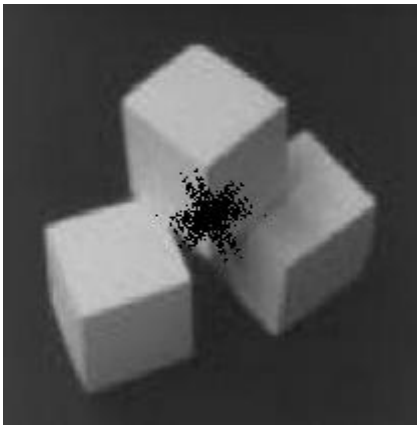
```
cout << "请输入保留的频率域: " << endl << "min: ";
cin >> min;
cout << "max: ";
cin >> max;

//滤波
IplImage* fourierResult = cvCloneImage(img);
cvSaveImage("FourierResultBefore.png", fourierResult);
for (int row = 0; row < fourierResult->height; row++)
{
    uchar* ptr = (uchar*)fourierResult->imageData + row*fourierResult->widthStep;
    for (int col = 0; col < fourierResult->width; col++)
    {
        int temp_frequency = sqrt(pow(frequency_value[row][col].real, 2) + pow(frequency_value[row][col].imag, 2));
        if (temp_frequency >= min && temp_frequency <= max)
        {
            //do nothing
        }
        else
        {
            ptr[col] = 0;//置为黑色
        }
    }
}
cvSaveImage("FourierResult.png", fourierResult);
cvReleaseImage(&fourierResult);
```

这里的 max 和 min 是输入的频率域。

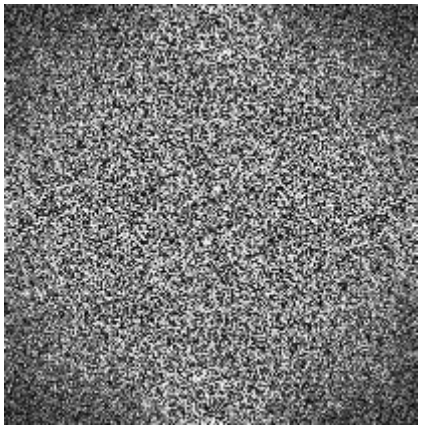
Part 3 output



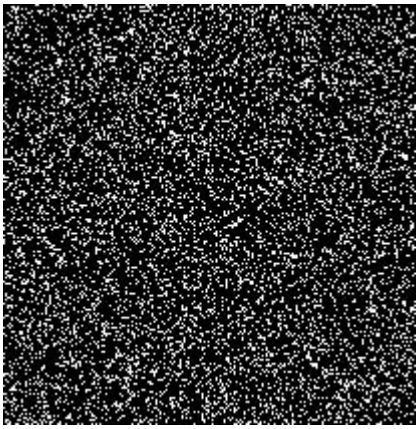
原图



滤波后



频谱图



相位谱图

Part 4 problems

虽然傅里叶变换是完全按照公式来计算，但是得到的频谱图跟网上的频谱图比起来差异较大，不知道是哪里有问题...

而且傅里叶变换在计算每个点时都要遍历整张图的信息，这样计算的次数就是 N^2，N 是图像中像素点的个数，导致运行时间较长。