WS 2019/20	Exercises Digital Image Processing	Solution Exercise No. 11

Image restauration in the frequency domain

Objective of exercises:

Allgemein

- Image restauration
- Filtering in the frequency domain

Solution for task 11.1a

Task: *Write a function showing the Fourier spectrum of an input image.*

The Fourier spectrum is computed by the equation listed below (1). It should be kept in mind that for the image to be displayed correctly it has to be adjusted to the spectrum and clipped afterwards.

$$|F(u,v)| = \sqrt{Re\{F(u,v)\}^2 + Im\{F(u,v)\}^2} \tag{1}$$

Determination of the interface:

```
// Generates the Fourier spectrum of a komplexwertigen input image <input>
// and stores it in a gray image <output>.
void viewFourierSpectrum (ComplexImage& input, GrayImage& output)
```

Implementation:

```
//
//
//
void viewFourierSpectrum (ComplexImage& input, GrayImage& output)
{
    int width  = input.getWidth();
    int height = input.getHeight();

    float*   odata = output.getData();
    Complex* idata = input.getData();

    for (int i=0; i<width*height; i++)
    {
        odata[i] = sqrt(idata[i].re * idata[i].re + idata[i].im * idata[i].im) * 255.0;
    }

    clip(output);
}

// ------------------------------------------------------------------------------

//
//
//
void buildFilterTransferFunction (GrayImage& mask, int d0, int n, int u0, int v0)
{
    int    width  = mask.getWidth();
    int    height = mask.getHeight();
    float* data   = mask.getData();

    int    m2     = width / 2;
    int    n2     = height / 2;

    for (int v=0; v<height; v++)
    {
        for (int u=0; u<width; u++)
        {
            float Duv1 = sqrt((double)((u-m2-u0)*(u-m2-u0) + (v-n2-v0)*(v-n2-v0)));
            float Duv2 = sqrt((double)((u-m2+u0)*(u-m2+u0) + (v-n2+v0)*(v-n2+v0)));

            data[v*width+u] = 1.0 / (1.0 + powf(((d0*d0)/(Duv1*Duv2)), n));
        }
    }

    saveFilterMask(mask, "filter.mask");

    scale(mask);
    mask.show();
    mask.save();
```

```
}

// ------------------------------------------------------------------------------

//
//
//
void doFilteringInFrequencyDomain (GrayImage& input, GrayImage& output)
{
    int width    = input.getWidth();
    int height   = input.getHeight();

    float* idata = input.getData();
    float* odata = output.getData();

    ComplexImage fuv = ComplexImage(width, height);
    ComplexImage tmp = ComplexImage(width, height);
    Complex* fuvdata = fuv.getData();
    Complex* tmpdata = tmp.getData();


    for (int v=0; v<height; v++)
    {
        for (int u=0; u<width; u++)
        {
            int index = v*width+u;
            fuvdata[index].re = idata[index] * pow(-1.0, u+v);
            fuvdata[index].im = 0;
        }
    }

    fourier_transform(fuv, tmp);

    GrayImage filter;
    loadFilterMask("filter.mask", filter);
    float* fdata = filter.getData();

    for (int i=0; i<input.getSize(); i++)
    {
        tmpdata[i].re = tmpdata[i].re * fdata[i];
        tmpdata[i].im = tmpdata[i].im * fdata[i];
    }

    inverse_fourier_transform(tmp, fuv);

    for (int v=0; v<height; v++)
    {
        for (int u=0; u<width; u++)
        {
            int index = v*width+u;
            odata[index] = fuvdata[index].re * pow(-1.0, u+v);
        }
    }
}

// ------------------------------------------------------------------------------

int main (int argc, char** argv)
{
    /* ********************************** */
    /* Exercise 12                        */
    /* ********************************** */

    cout << "Bildwiederherstellung - Bitte Funktion wählen" << endl;
    cout << "1 - Originalbild mit Wellenmuster verfremden (kx < 1 & ky < 1) und speichern" << endl;
    cout << "2 - Fourierspektrum von Bild anzeigen" << endl;
    cout << "3 - Filter passend zum Fourierspektrum erstellen und speichern" << endl;
    cout << "4 - Gestörtes Bild filtern und Ergebnisanzeigen" << endl;

    int choice;
    cin >> choice;

    GrayImage image;
    image.load();

    int     width  = image.getWidth();
    int     height = image.getHeight();
```

```
GrayImage output(width, height);

float* idata  = image.getData();
float* odata  = output.getData();

switch (choice)
{
    case 1:
    {
        float kx, ky;

        cout << "kx: ";
        cin >> kx;
        cout << "ky: ";
        cin >> ky;

        wellenmuster (output, kx, ky, 0);
        output.show();

        for (int i=0; i<image.getSize(); i++)
        {
            idata[i] = idata[i] + idata[i] * ((odata[i] - 127.5) / 512.0);
        }

        clip(image);
        image.show();
        image.save();

        break;
    }
    case 2:
    {
        ComplexImage fuv(width, height);
        ComplexImage tmp(width, height);
        GrayImage output(width, height);

        fuv.copy(image);

        fourier_center(fuv);
        fourier_transform(fuv, tmp);

        viewFourierSpectrum(tmp, output);

        output.show();
        output.save();

        break;
    }
    case 3:
    {
        if (width  == 0) width = 512;
        if (height == 0) height = 512;

        GrayImage filter(width, height);

        int d0, u0, v0, n;

        cout << "Cut-Off Frequenz d0: ";
        cin >> d0;
        cout << "Ordnung n          : ";
        cin >> n;
        cout << "Koordinate u0      : ";
        cin >> u0;
        cout << "Koordinate v0      : ";
        cin >> v0;

        buildFilterTransferFunction(filter , d0, n, u0, v0);

        break;
    }
    case 4:
    {
        doFilteringInFrequencyDomain(image, output);

        scale(output);
        output.show();
```

```
            output.save();

            break;
        }
    }
    cout << "FINISHED.\n";
    return 1;
}
```

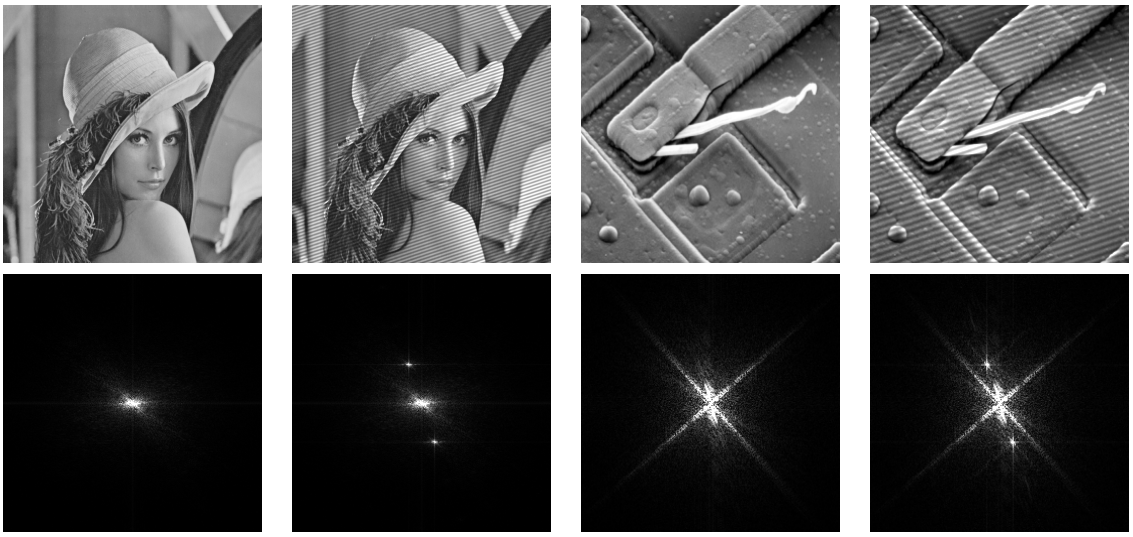

Figure 1: oben: Original images (with and without wave), below: Fourier spectra of these images