6. Low Pass Filter and High Pass Filter

1. Low Pass Filter (lena):

- Ideal Low Pass Filter:

Algorithm:

The image is first transformed by **DFT** to obtain a complex array of each pixel. The **real** and **imaginary** parts of each complex number are multiplied by $H(u,v)$ respectively. The formula of $H(u,v)$ is:

$$H(u,v)=\begin{cases}1, D(u,v)\le D_0\\ 0, D(u,v)>D_0\end{cases}$$

where D_0 represents the radius of the passband. The calculation method of $D(u,v)$ is also the distance between two points, through the formula:

$$D(u,v)=\sqrt{\left(u-\frac{P}{2}\right)^2+\left(v-\frac{Q}{2}\right)^2}$$

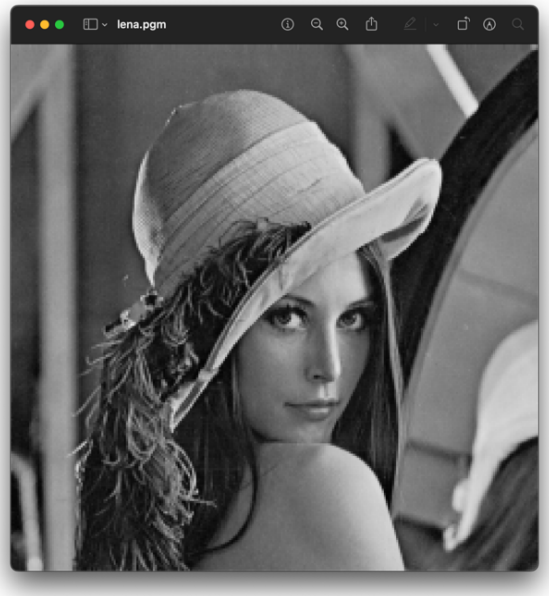
Where, P and Q are the length and width of the image.

Finally, use **iDFT** to restore the processed image.

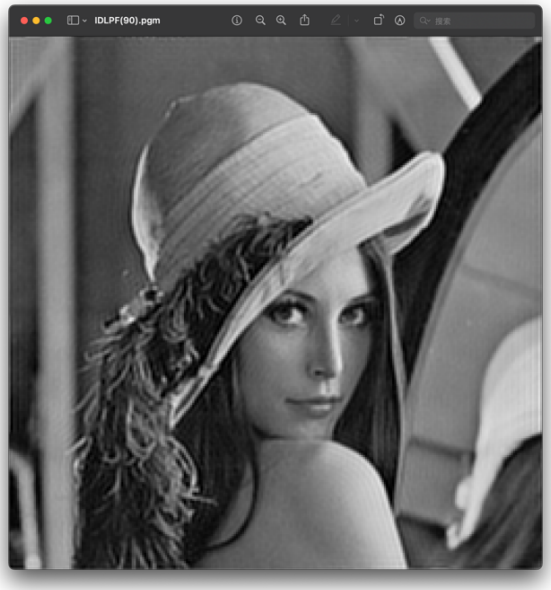
Results (including pictures):

Process result of "lena.pgm":

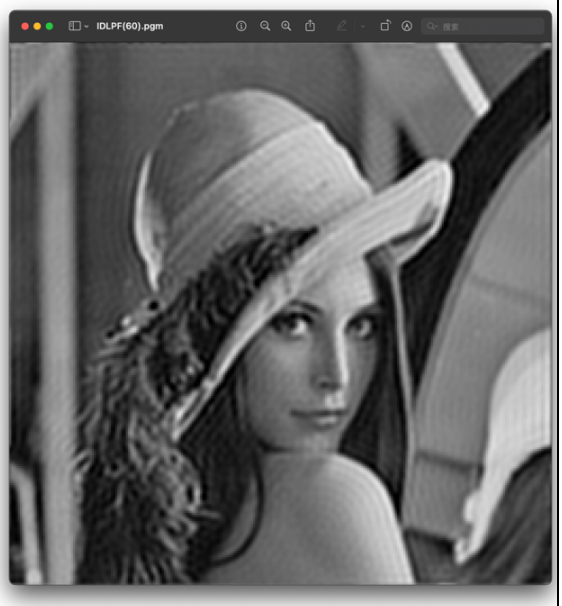
Source Image:



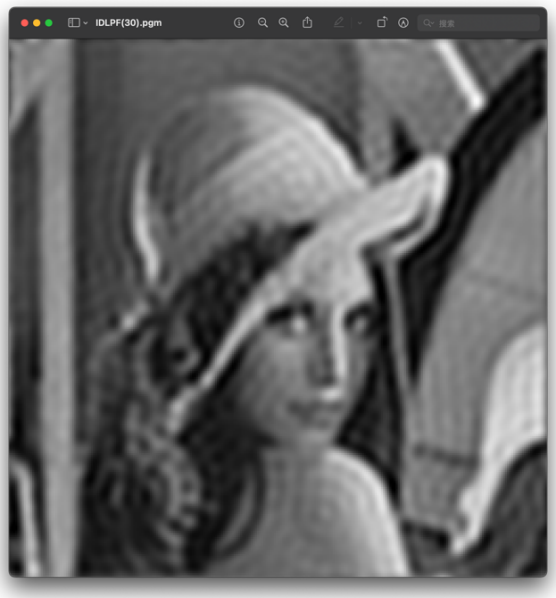
Result of IDLPF($D_0 = 90$):



Result of IDLPF($D_0 = 60$):



Result of IDLPF($D_0 = 30$):



Discussion:

The ideal low-pass filter can completely reject all frequency signals above the cut-off frequency and the signals below the cut-off frequency can pass unaffected. Since the transition characteristics of the ideal low-pass filter are too sharp, ringing will occur. It filters out high frequency components, so it blurs the image. In addition, the smaller the value of D_0 is chosen, the more obvious the effect of image processing will be, that is, the more blurred it will be.

Codes:

```
104 void IDLPF(Image *image, float *real_array, float *imaginary_array) {
105     unsigned char *tempin, *tempout;
106     float height = image->Height, width = image->Width;
107     float real[(int)(height*width)], imaginary[(int)(height*width)];
108     Image *outimage;
109     outimage = CreateNewImage(image, (char*)"#testing function");
110     tempin = image->data;
111     tempout = outimage->data;
112 
113     for(int i = 0; i < height; i++) {
114         for(int j = 0; j < width; j++) {
115             float dis = sqrt(pow((float)i - height/2, 2) + pow((float)j - width/2, 2));
116             int H;
117             if(dis <= 60) H = 1;
118             else H = 0;
119             real[(int)(i*height + j)] = real_array[(int)(i*height + j)] * H;
120             imaginary[(int)(i*height + j)] = imaginary_array[(int)(i*height + j)] * H;
121         }
122     }
123     printf("IDLPF Finished!\n");
124     outimage = iDFT(image, real, imaginary);
125     SavePNMImage(outimage, (char*)"IDLPF.pgm");
126 }
```

- Butterworth Low Pass Filter:

Algorithm:

The only difference compared to IDLFT is that it changes the formula of $H(u, v)$:

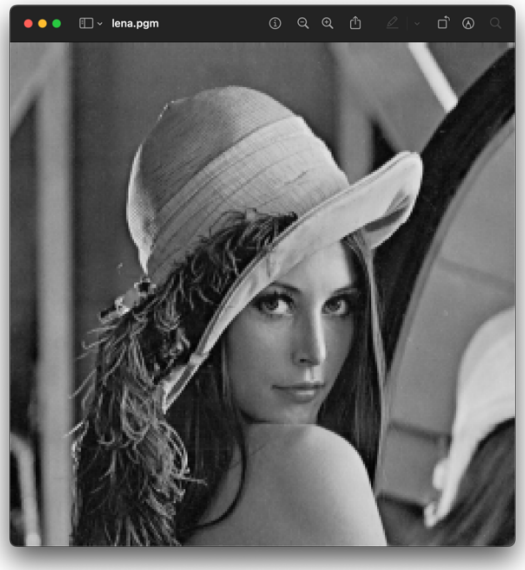
$$H(u,v)=\frac{1}{1+\left(D(u,v)/D_0\right)^{2n}}$$

where n is the order of the Butterworth filter.

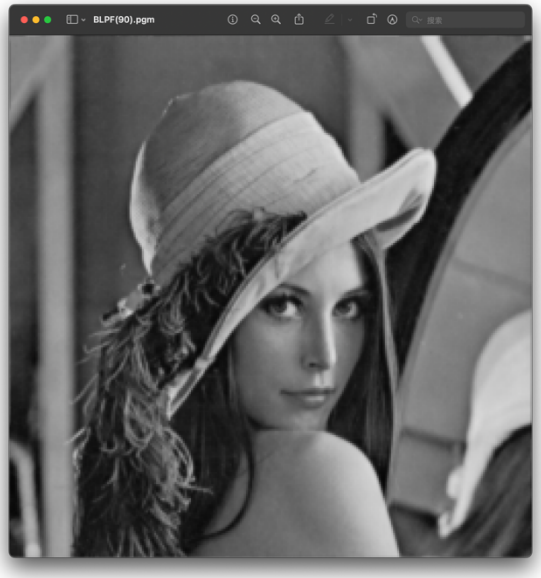
Results (including pictures):

Process result of "lena.pgm":

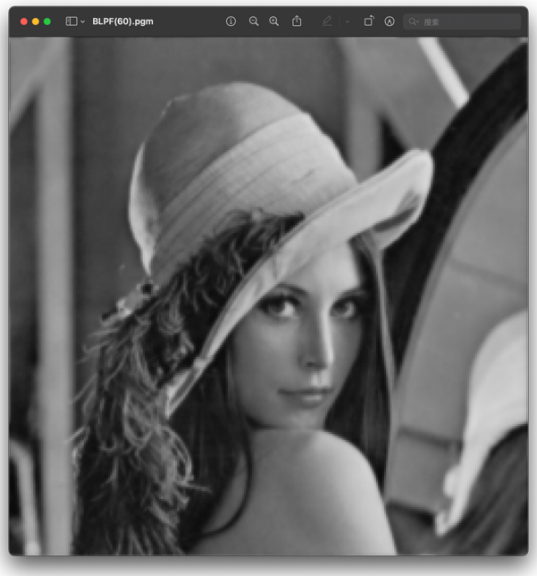
Source Image:



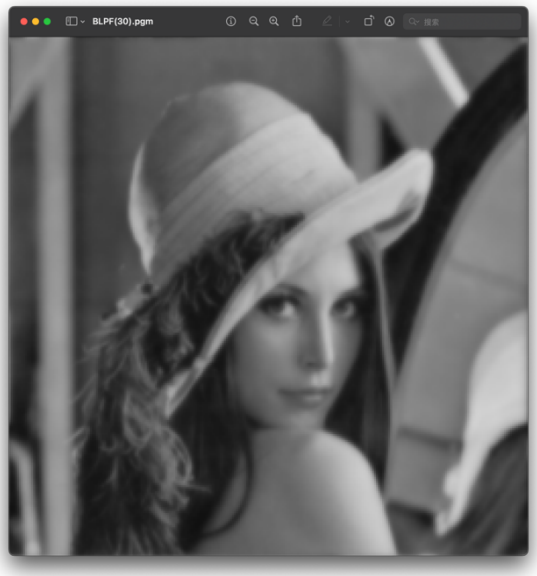
Result of BLPF($D_0 = 90, n = 1$):



Result of BLPF($D_0 = 60, n = 1$):



Result of BLPF($D_0 = 30, n = 1$):



Discussion:

According to the characteristics of the Butterworth filter, the frequency response curve in the pass band is flat and has no ripple, while it gradually drops to zero in the stop band. So the image it processes is significantly more natural and smoother than the IDLPF. And smaller the value of D_0 is chosen, the more blurred the image will be. However, as the number of times n increases, the ringing phenomenon will become more obvious.

Codes:

```
void BLPF(Image *image, float *real_array, float *imaginary_array) {
    unsigned char *tempin, *tempout;
    float height = image->Height, width = image->Width;
    float real[(int)(height*width)], imaginary[(int)(height*width)];
    Image *outimage;
    outimage = CreateNewImage(image, (char*)"#testing function");
    tempin = image->data;
    tempout = outimage->data;

    for(int i = 0; i < height; i++) {
        for(int j = 0; j < width; j++) {
            float dis = sqrt(pow((float)i - height/2, 2) + pow((float)j - width/2, 2));
            float H = 1 / (1 + pow(dis / (float)60, 2));
            real[(int)(i*height + j)] = real_array[(int)(i*height + j)] * H;
            imaginary[(int)(i*height + j)] = imaginary_array[(int)(i*height + j)] * H;
        }
    }
    printf("BLPF Finished!\n");
    outimage = iDFT(image, real, imaginary);
    SavePNMImage(outimage, (char*)"BLPF.pgm");
}
```

- Gaussian Low Pass Filter:

Algorithm:

The formula of $H(u,v)$ is that:

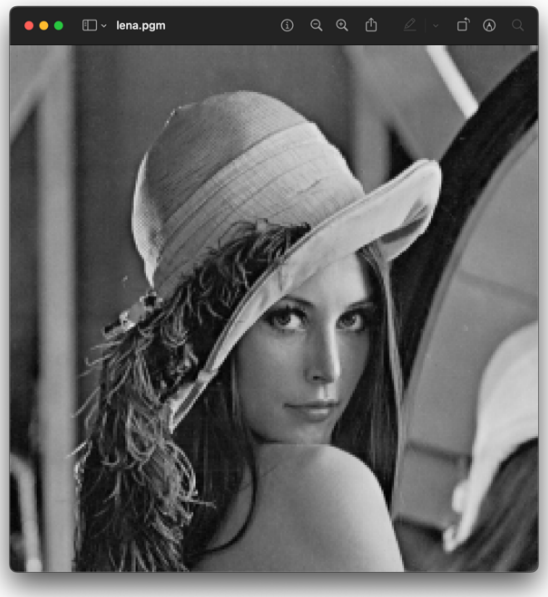
$$H(u,v) = e^{\frac{-D^2(u,v)}{2D_0^2}},$$

And the other parts are same as the previous algorithms.

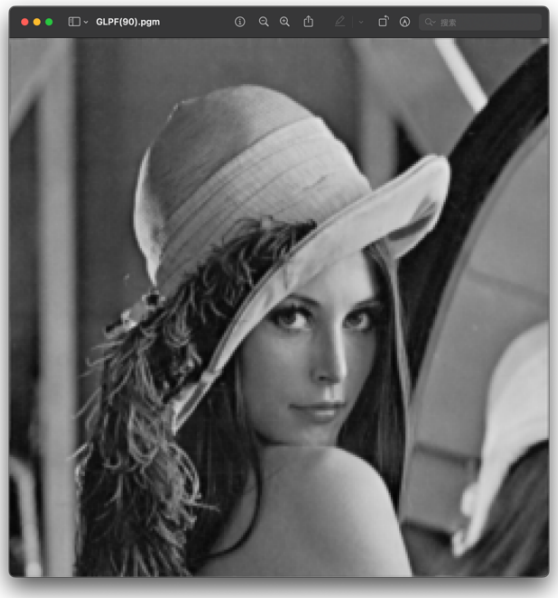
Results (including pictures):

Process result of "lena.pgm":

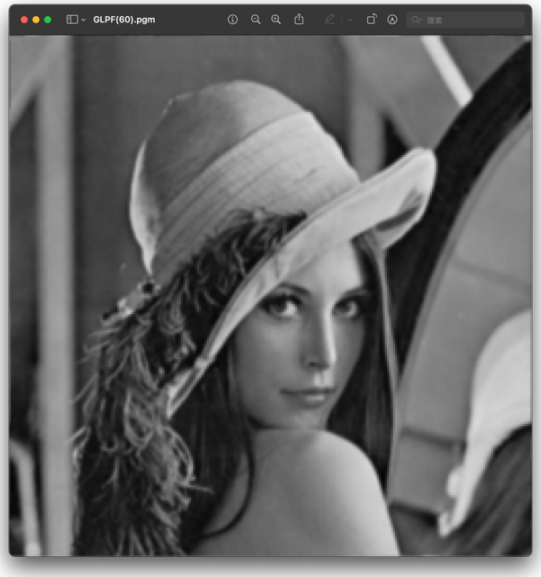
Source Image:



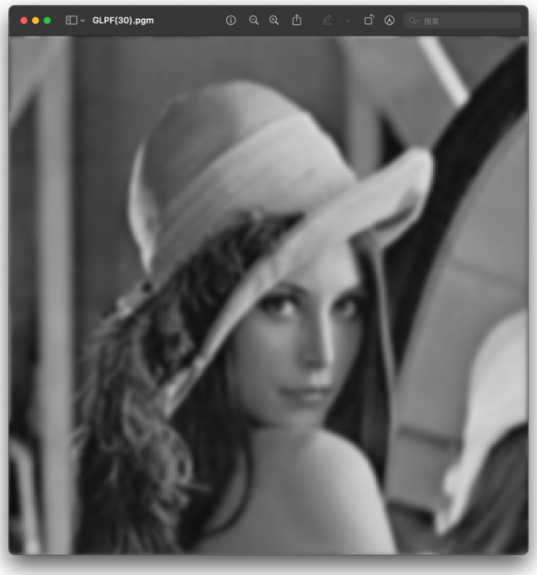
Result of GLPF($D_0 = 90$):



Result of GLPF($D_0 = 60$):



Result of GLPF($D_0 = 30$):



Discussion:

The transition characteristics of the Gaussian filter are very flat, so there is no ringing even D_0 is very small. The processed images are very smooth, and smaller the value of D_0 is chosen, the more blurred the image will be. Moreover, this algorithm is likely to be suitable for removing image noise.

Codes:

```
void GLPF(Image *image, float *real_array, float *imaginary_array) {
    unsigned char *tempin, *tempout;
    float height = image->Height, width = image->Width;
    float real[(int)(height*width)], imaginary[(int)(height*width)];
    Image *outimage;
    outimage = CreateNewImage(image, (char*)"#testing function");
    tempin = image->data;
    tempout = outimage->data;

    for(int i = 0; i < height; i++) {
        for(int j = 0; j < width; j++) {
            float dis = sqrt(pow((float)i - height/2, 2) + pow((float)j - width/2, 2));
            float H = pow(M_E, pow(dis, 2) * -1 / (pow(60, 2) * 2));
            real[(int)(i*height + j)] = real_array[(int)(i*height + j)] * H;
            imaginary[(int)(i*height + j)] = imaginary_array[(int)(i*height + j)] * H;
        }
    }
    printf("GLPF Finished!\n");
    outimage = iDFT(image, real, imaginary);
    SavePNMImage(outimage, (char*)"GLPF.pgm");
}
```

2. High Pass Filter (fingerprint1, fingerprint2):

Algorithm:

The image is first transformed by **DFT** to obtain a complex array of each pixel. The **real** and **imaginary** parts of each complex number are multiplied by $H(u,v)$ respectively. The formula of $H(u,v)$ is: (For the effect of image sharpening, the **BHPF** is selected here)

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{2n}}$$

where D_0 represents the radius of the passband. The calculation method of $D(u,v)$ is also the distance between two points, through the formula:

$$D(u,v) = \sqrt{\left(u - \frac{P}{2}\right)^2 + \left(v - \frac{Q}{2}\right)^2}$$

Where, P and Q are the length and width of the image.

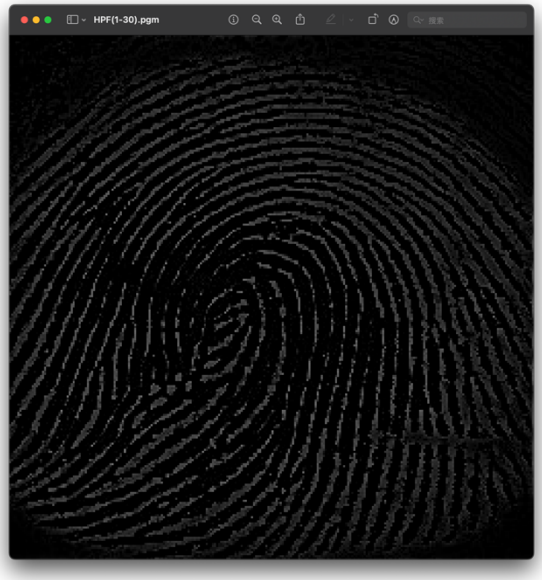
Finally, use **iDFT** to restore the processed image.

Results (including pictures):

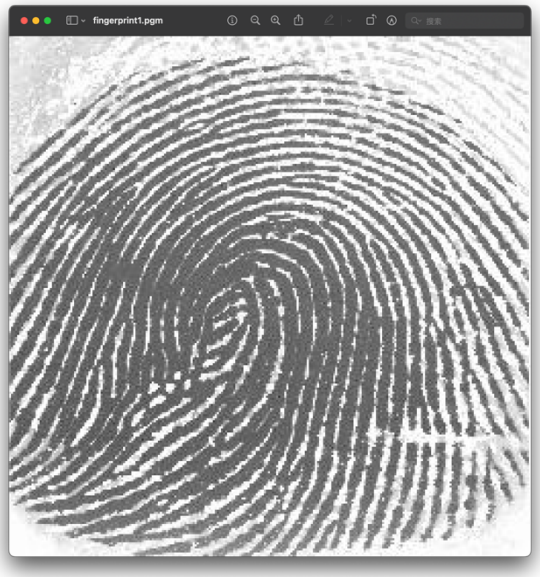
Process result of "fingerprint1.pgm":

Source Image:

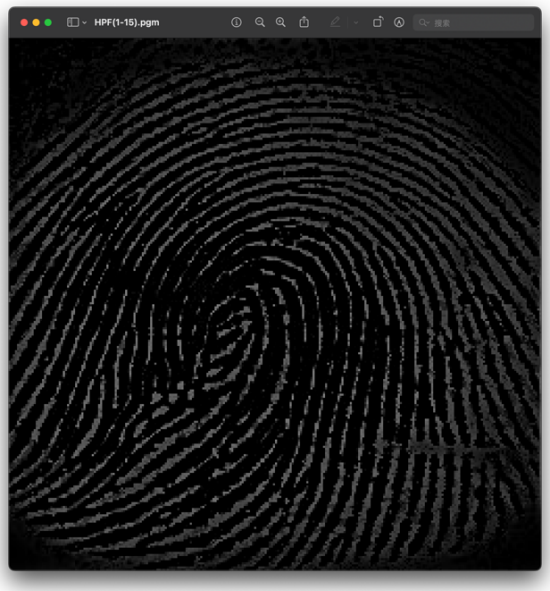
Result of BHPF($D_0 = 30$):



Source Image:



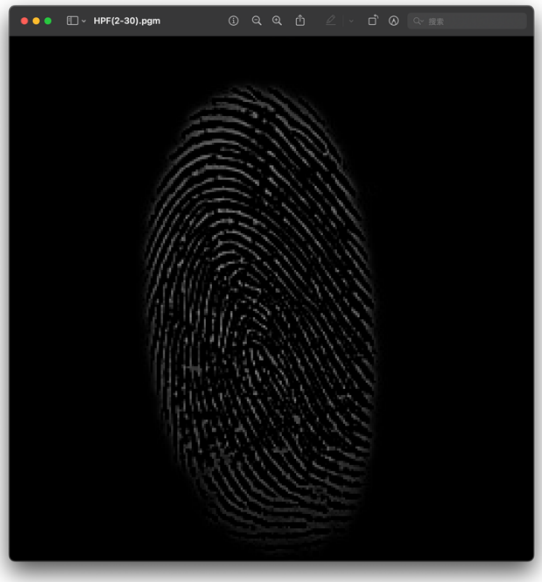
Result of BHPF($D_0 = 15$):



Process result of “fingerprint2.pgm”

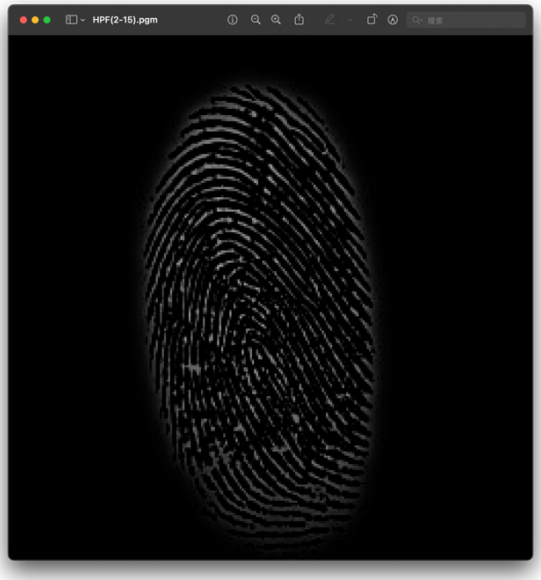
Source Image:

Result of BHPF($D_0 = 30$):



Source Image:

Result of BHPF($D_0 = 15$):



Discussion:

For a more obvious sharpening effect I chose the **Butterworth high pass filter**. As we can see from the output results, the values of high frequencies are removed and only the low frequency values are preserved, so the edges of the fingerprints are sharpened and highlighted. And the lower the D_0 is chosen, the more obvious the effect of image processing will be.

Codes:

```
172 void HPF(Image *image, float *real_array, float *imaginary_array) {
173     unsigned char *tempin, *tempout;
174     float height = image->Height, width = image->Width;
175     float real[(int)(height*width)], imaginary[(int)(height*width)];
176     Image *outimage;
177     outimage = CreateNewImage(image, (char*)"#testing function");
178     tempin = image->data;
179     tempout = outimage->data;
180 
181     for(int i = 0; i < height; i++) {
182         for(int j = 0; j < width; j++) {
183             float dis = sqrt(pow((float)i - height/2, 2) + pow((float)j - width/2, 2));
184             float H = 1 / (1 + pow((float)30 / dis, 2));
185             real[(int)(i*height + j)] = real_array[(int)(i*height + j)] * H;
186             imaginary[(int)(i*height + j)] = imaginary_array[(int)(i*height + j)] * H;
187         }
188     }
189     printf("HPF Finished!\n");
190     outimage = iDFT(image, real, imaginary);
191     SavePNMImage(outimage, (char*)"HPF.pgm");
192 }
```