Class Lab 6

Use two images for each operation to do the following operations and write down their advantages and disadvantages and explain your results:

1. ILPF (lena, camera):

Algorithm:

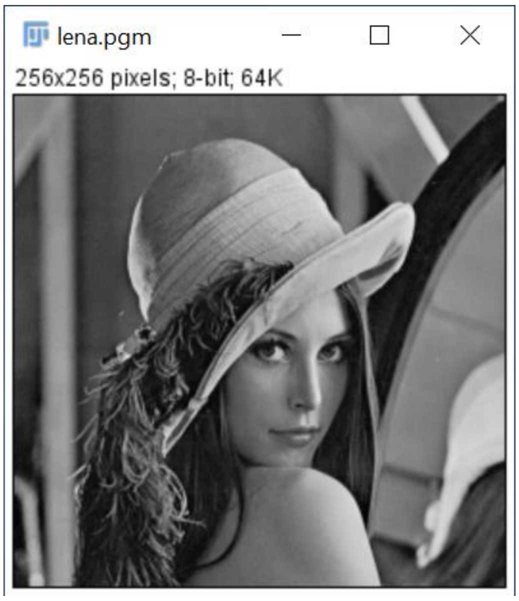
$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \le D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$$

$$D(u,v) = \left[(u - P/2)^2 + (v - Q/2)^2 \right]^{1/2}$$

Results (including pictures):

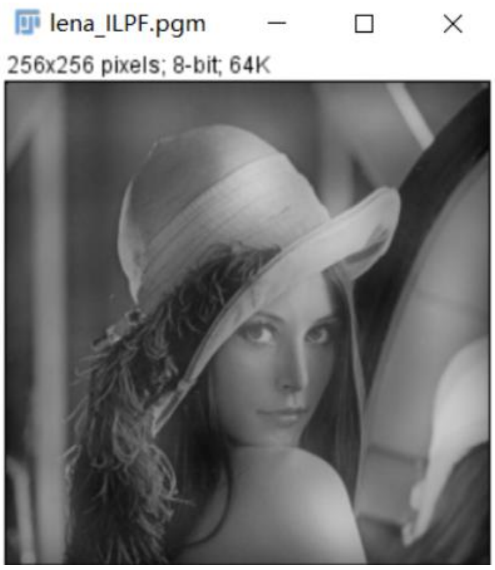
Result of processing “Lena.pgm”:

Source Image:



lena.pgm
256x256 pixels; 8-bit; 64K

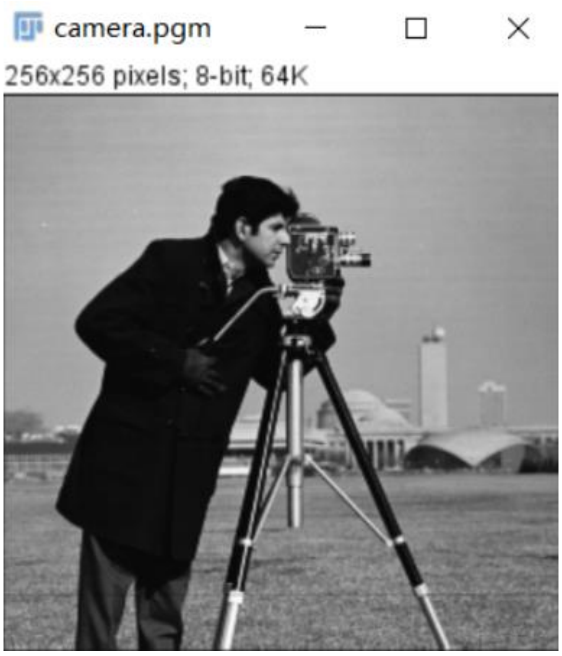
Result after ILPF:



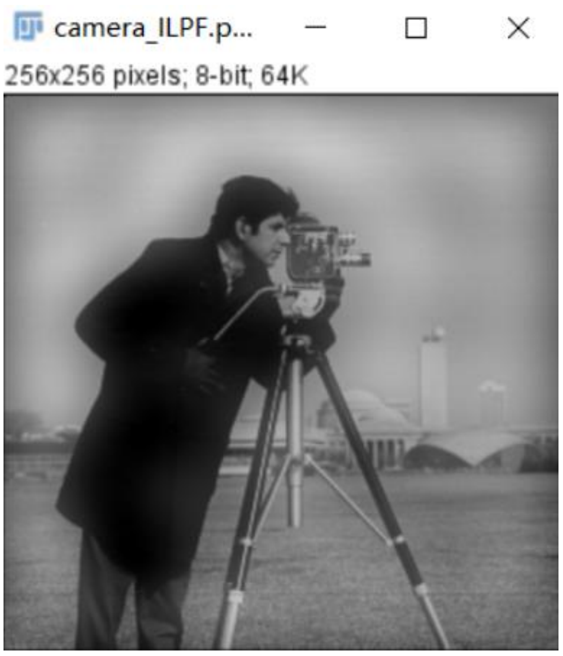
lena_ILPF.pgm
256x256 pixels; 8-bit; 64K

Result of processing “camera.pgm”:

Source Image:



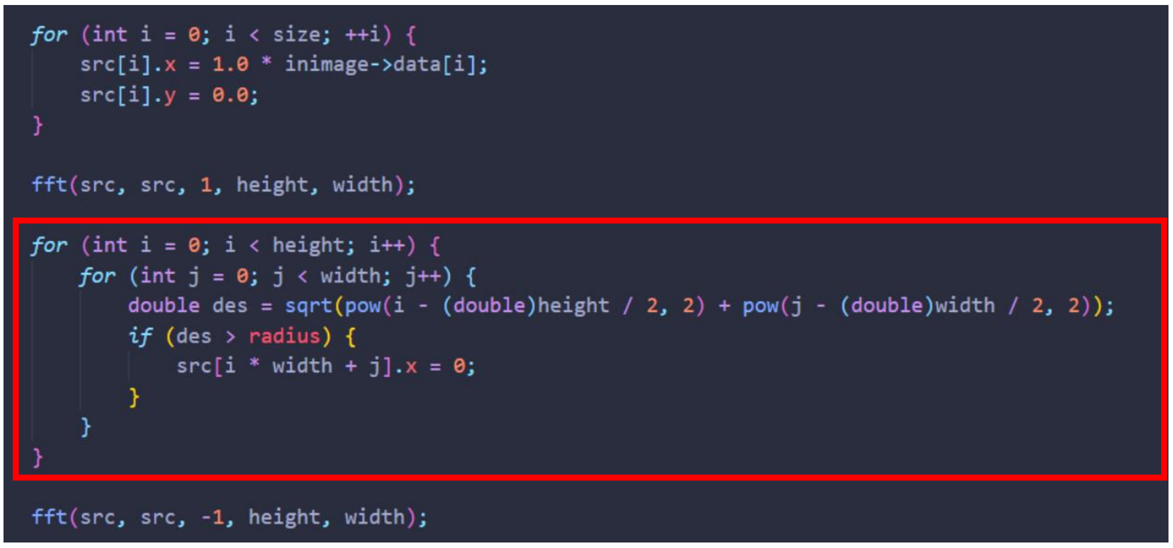
Result after ILPF:



Discussion:

The figure shows the results of ILPF. When the cutoff frequency is too small, it is not useful for practical purposes, unless the purpose of blurring is to eliminate all detail from the image. As the radius of filter increases, the power of filter becomes less and less, resulting in weaker and weaker blur. And even when only a small amount of information is filtered out, the ringing effect is obvious.

Codes:



```
for (int i = 0; i < size; ++i) {
    src[i].x = 1.0 * inimage->data[i];
    src[i].y = 0.0;
}

fft(src, src, 1, height, width);

for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        double des = sqrt(pow(i - (double)height / 2, 2) + pow(j - (double)width / 2, 2));
        if (des > radius) {
            src[i * width + j].x = 0;
        }
    }
}

fft(src, src, -1, height, width);
```

2. BLPF (lena, camera):

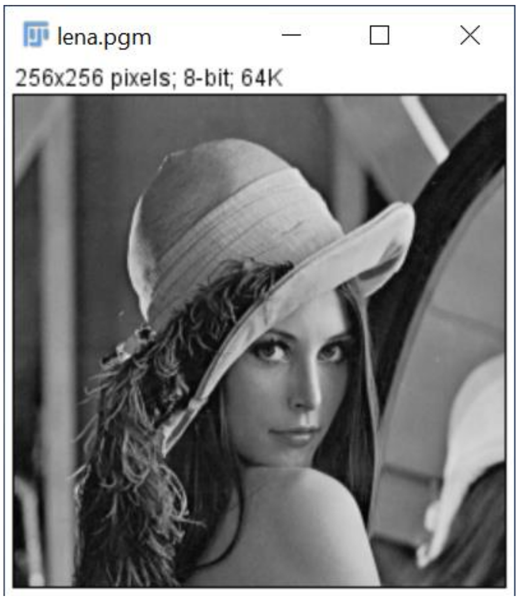
Algorithm:

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

Results (including pictures):

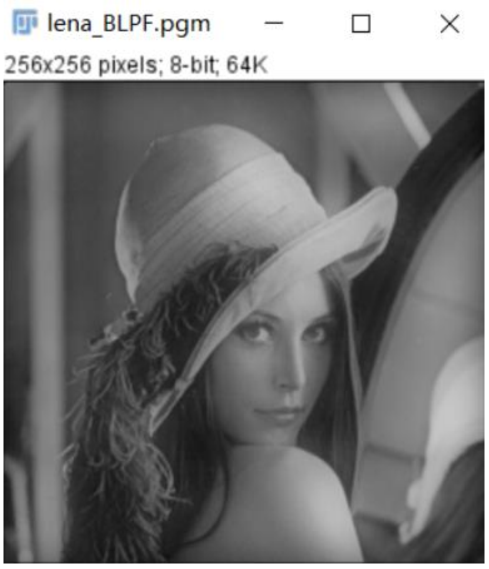
Result of processing "Lena.pgm":

Source Image:


lena.pgm

256x256 pixels; 8-bit; 64K

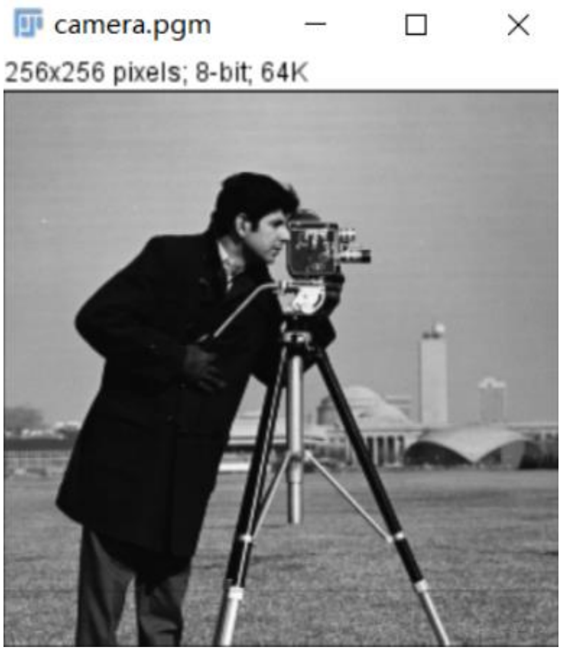
Result after BLPF:


lena_BLPF.pgm

256x256 pixels; 8-bit; 64K

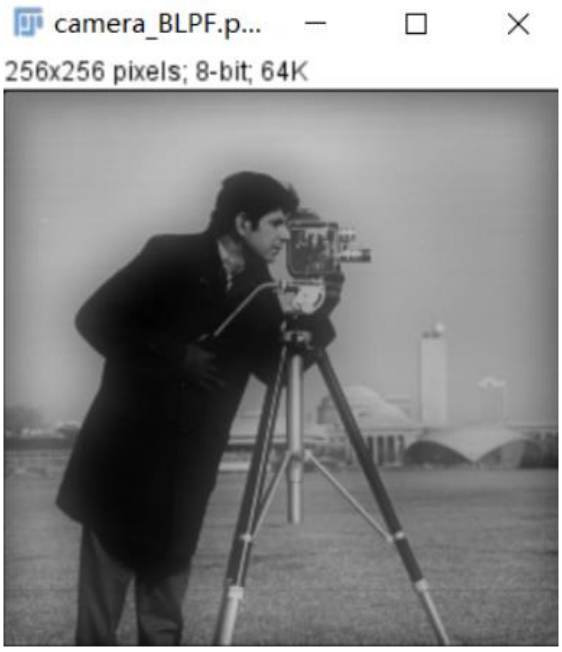
Result of processing "camera.pgm":

Source Image:


camera.pgm

256x256 pixels; 8-bit; 64K

Result after BLPF:

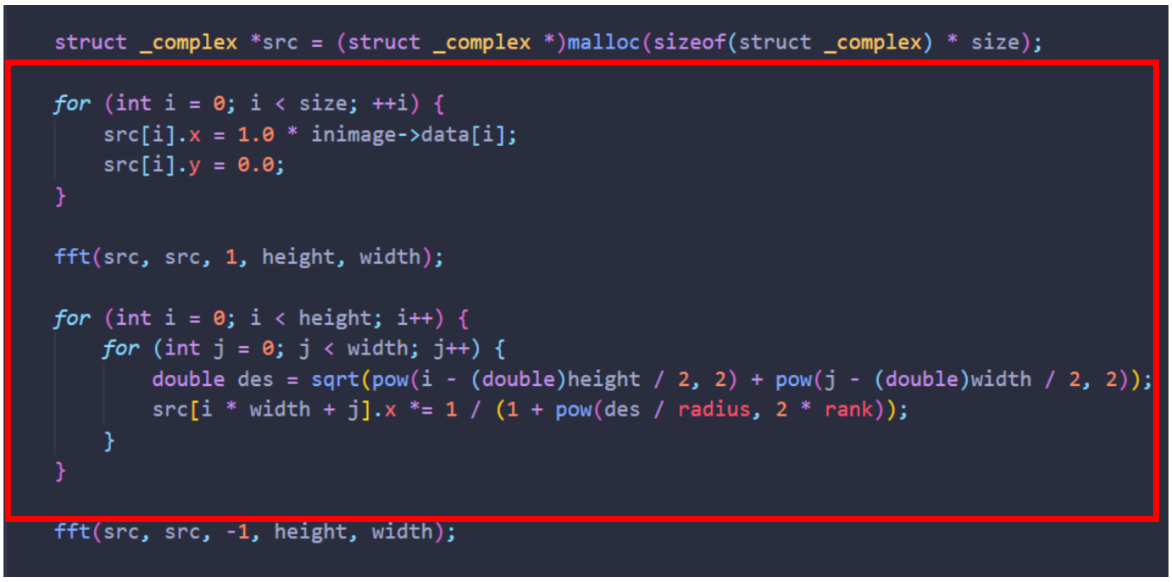

camera_BLPF.p...

256x256 pixels; 8-bit; 64K

Discussion:

The figure shows the BLPF results. The first order Butterworth filter in spatial domain has no ringing effect. Ringing effect is usually imperceptible in order 2 and 3 filters, but it is obvious in higher order filters.

Codes:



```
struct _complex *src = (struct _complex *)malloc(sizeof(struct _complex) * size);

for (int i = 0; i < size; ++i) {
    src[i].x = 1.0 * inimage->data[i];
    src[i].y = 0.0;
}

fft(src, src, 1, height, width);

for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        double des = sqrt(pow(i - (double)height / 2, 2) + pow(j - (double)width / 2, 2));
        src[i * width + j].x *= 1 / (1 + pow(des / radius, 2 * rank));
    }
}

fft(src, src, -1, height, width);
```

3. GLPF (lena, camera):

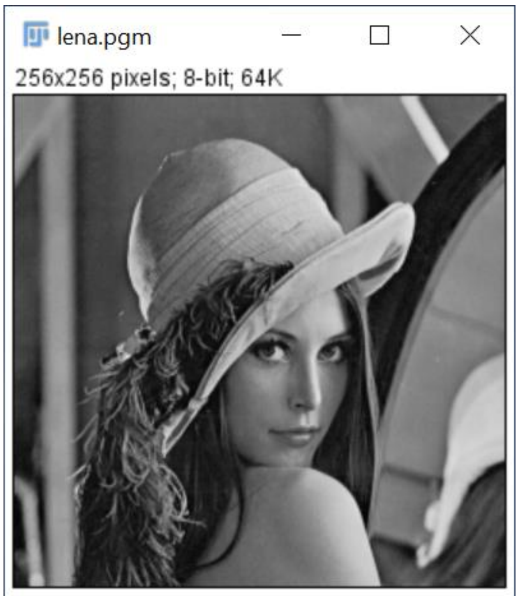
Algorithm:

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

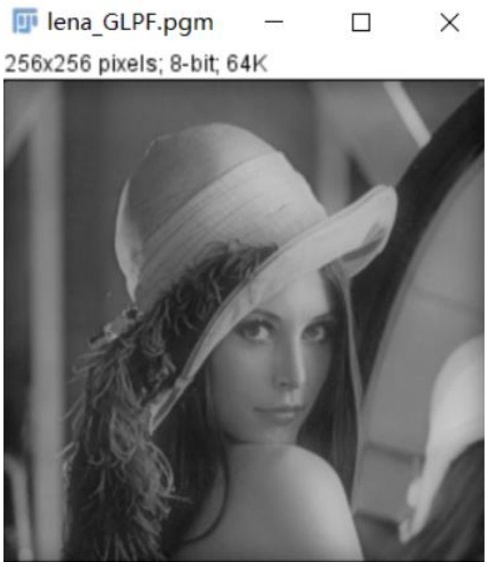
Results (including pictures):

Result of processing “Lena.pgm”:

Source Image:

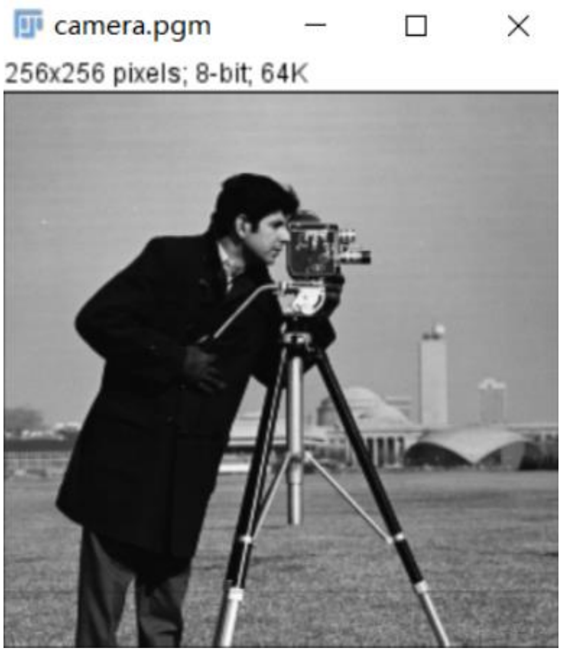


Result after GLPF:

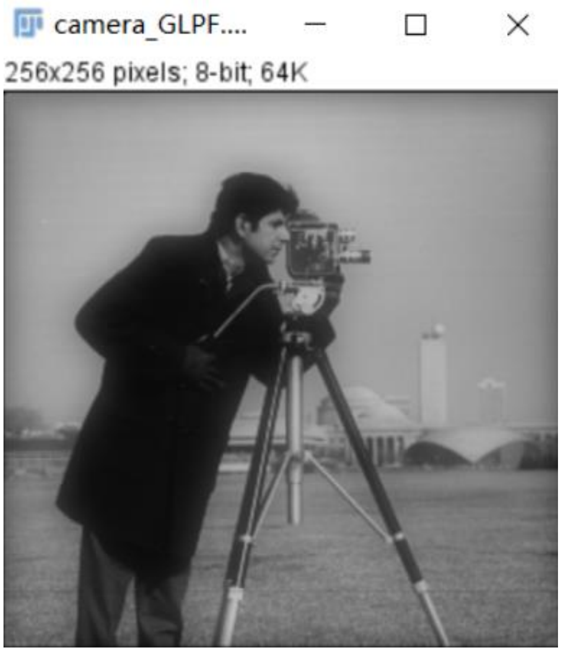


Result of processing “camera.pgm”:

Source Image:



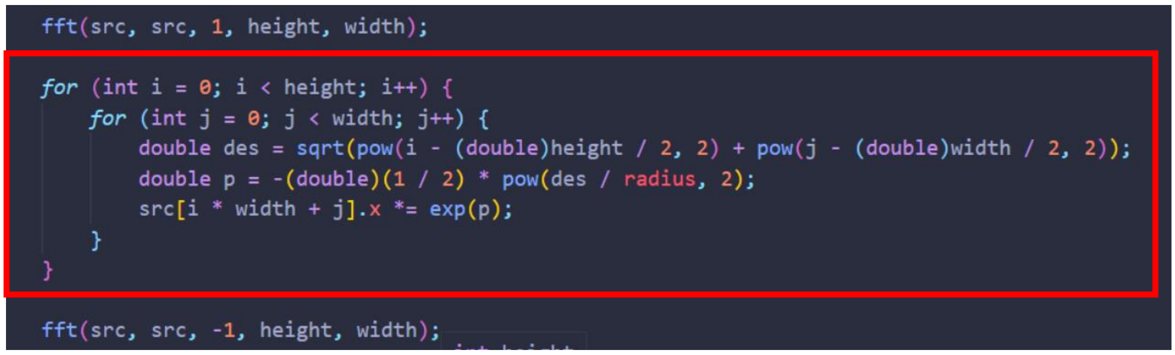
Result after GLPF:



Discussion:

Compared to the results obtained with an ILPF, we note a smooth transition in blurring as a function of increasing cutoff frequency. The GLPF achieved slightly less smoothing than the ILPF. The key difference is that we are assured of no ringing when using a GLPF.

Codes:



```
fft(src, src, 1, height, width);

for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        double des = sqrt(pow(i - (double)height / 2, 2) + pow(j - (double)width / 2, 2));
        double p = -(double)(1 / 2) * pow(des / radius, 2);
        src[i * width + j].x *= exp(p);
    }
}

fft(src, src, -1, height, width);
```

4. HPF and Thresholding (Fingerprint1, Fingerprint2):

Algorithm:

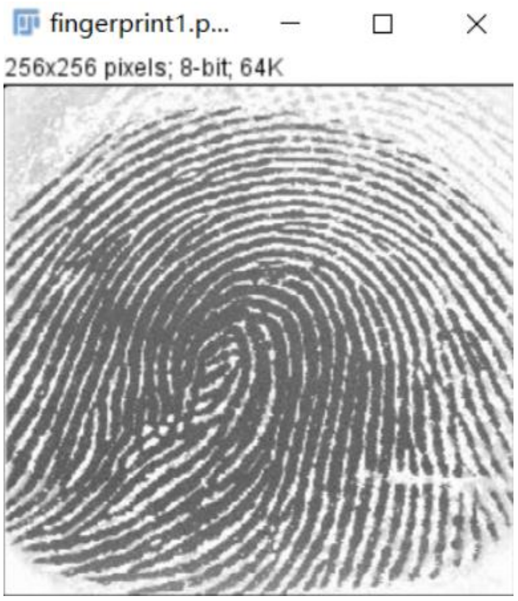
$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

$$f(x,y) = \begin{cases} 255 & f(x,y) > 100 \\ 0 & f(x,y) \leq 100 \end{cases}$$

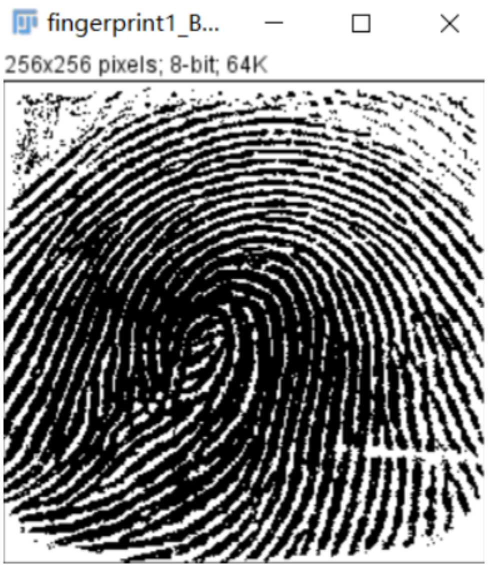
Results (including pictures):

Result of processing "Fingerprint1.pgm":

Source Image:

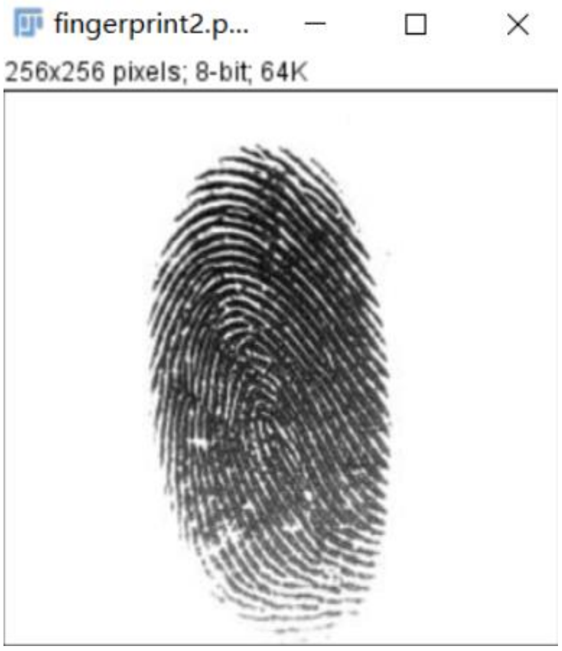


Result after HPF_T:

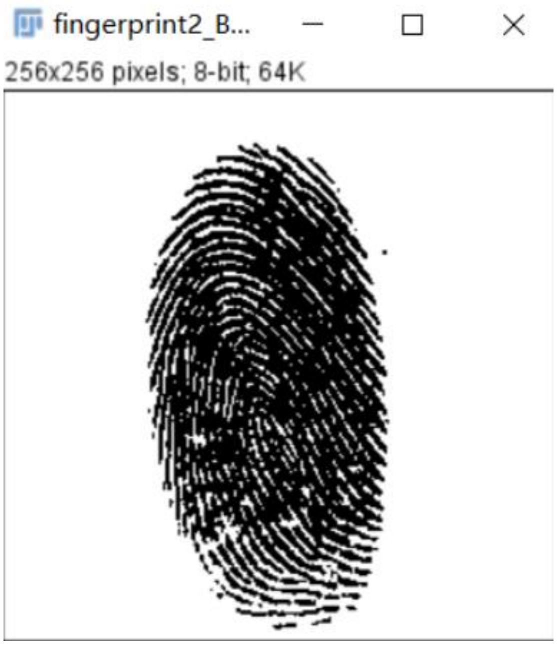


Result of processing "Fingerprint2.pgm":

Source Image:



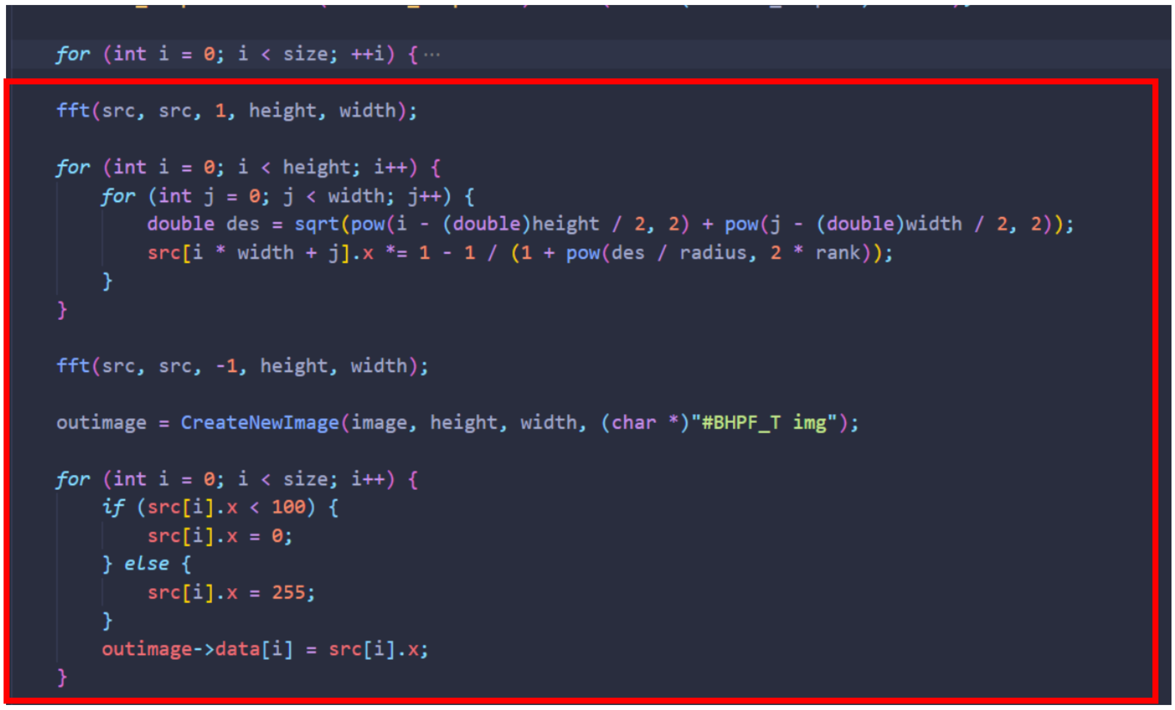
Result after GLPF:



Discussion:

This is the result of using a Butterworth highpass filter of order 4 with a cutoff frequency of 50. A fourth-order filter provides a sharp (but smooth) transition from low to high frequencies, with filtering characteristics between an ideal and a Gaussian filter. The idea is for D_0 to be close to the origin so that low frequencies are attenuated but not completely eliminated, so that tonality differences between the ridges and background are not lost completely. Choosing a value for D_0 between 5% and 10% of the long dimension of the image is a good starting point.

Codes:



```
for (int i = 0; i < size; ++i) {⋯

fft(src, src, 1, height, width);

for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        double des = sqrt(pow(i - (double)height / 2, 2) + pow(j - (double)width / 2, 2));
        src[i * width + j].x *= 1 - 1 / (1 + pow(des / radius, 2 * rank));
    }
}

fft(src, src, -1, height, width);

outimage = CreateNewImage(image, height, width, (char *)"#BHPF_T img");

for (int i = 0; i < size; i++) {
    if (src[i].x < 100) {
        src[i].x = 0;
    } else {
        src[i].x = 255;
    }
    outimage->data[i] = src[i].x;
}
```