10장 주파수 영역 처리

푸리에 변환

- 1D Fourier Transform

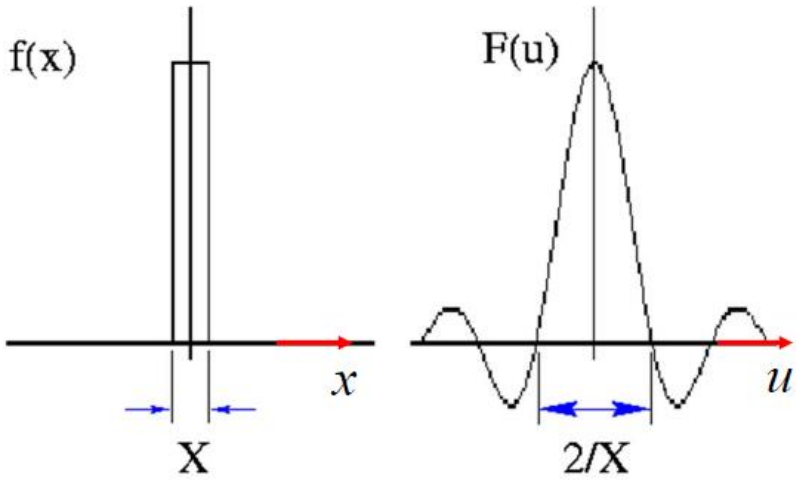
$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux}\, dx,$$
$$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux}\, du$$

x : time, u : frequency

- Example

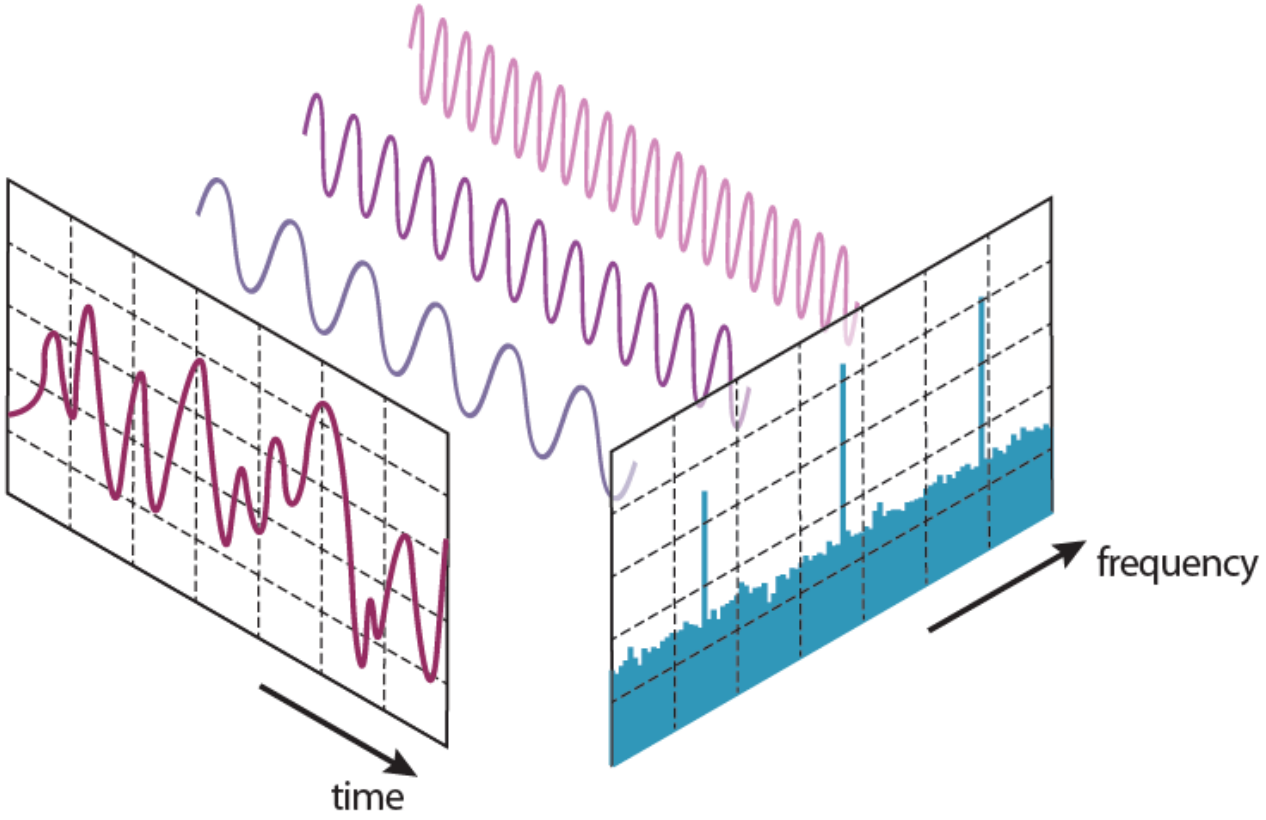
$$f(x) = \begin{cases} 1, & |x| < \frac{X}{2}, \\ 0, & |x| \geq \frac{X}{2}. \end{cases}$$

$$\begin{aligned} F(u) &= \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux} dx \\ &= \int_{-X/2}^{X/2} e^{-j2\pi ux} dx \\ &= \frac{1}{-j2\pi u}[e^{-j2\pi uX/2} - e^{j2\pi uX/2}] \\ &= X\frac{\sin(\pi Xu)}{(\pi Xu)} = X\mathrm{sinc}(\pi \mathrm{Xu}). \end{aligned}$$



푸리에 변환

- 1D Fourier Transform



푸리에 변환

- 2D Fourier Transform

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(ux+vy)}\,dx\,dy,$$
$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{j2\pi(ux+vy)}\,du\,dv$$

(x, y) : spatial domain
(u, v) : frequency domain

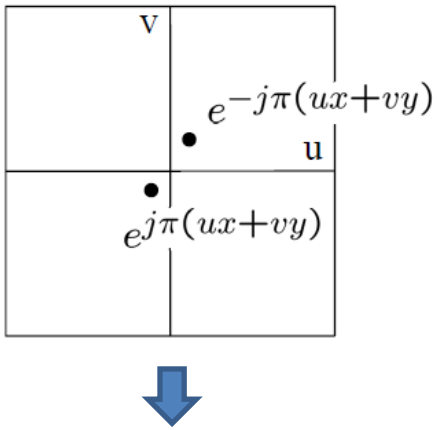
- Properties

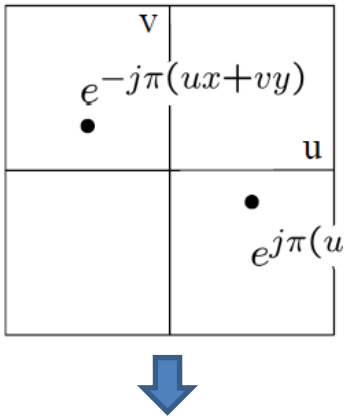
$$F(u,v) = F_{\mathrm{R}}(u,v) + jF_{\mathrm{I}}(u,v)$$

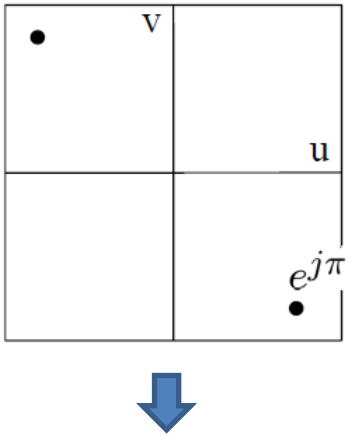
- $|F(u,v)|$ is the magnitude spectrum
- $\arctan(F_{\mathrm{I}}(u,v)/F_{\mathrm{R}}(u,v))$ is the phase angle spectrum.
- Conjugacy: $f^*(x,y) \Leftrightarrow F(-u,-v)$
- Symmetry: $f(x,y)$ is even if $f(x,y) = f(-x,-y)$

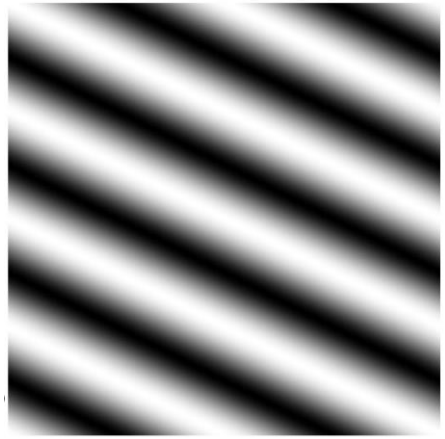
푸리에 변환

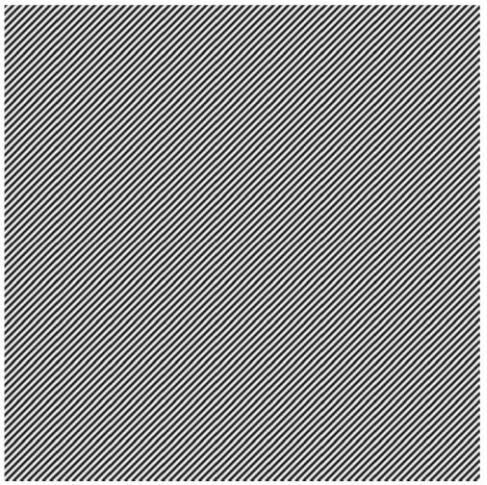
- $F(u,v) \Rightarrow f(x,y)$ examples







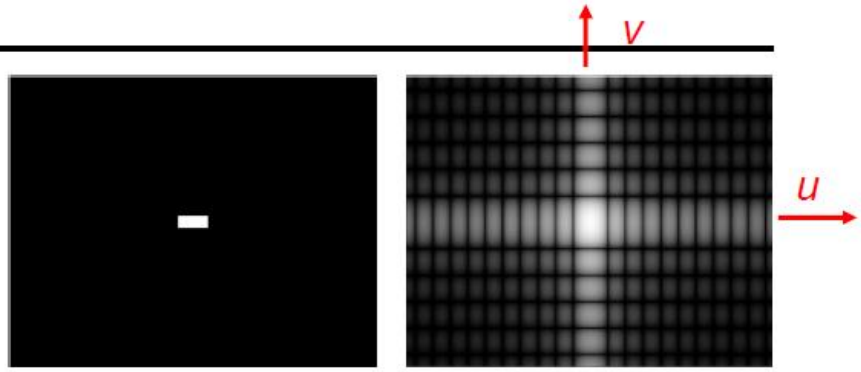




푸리에 변환

FT pair example 1

rectangle centred at origin
with sides of length X and Y



$$F(u,v) = \int\int f(x,y)e^{-j2\pi(ux+vy)}dxdy,$$

$$= \int_{-X/2}^{X/2} e^{-j2\pi ux}dx \int_{-Y/2}^{Y/2} e^{-j2\pi vy}dy, \quad \text{separability}$$

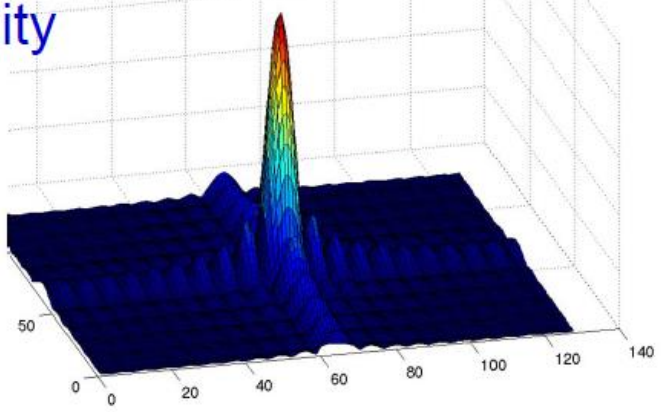
$$= \left[\frac{e^{-j2\pi ux}}{-j2\pi u}\right]_{-X/2}^{X/2} \left[\frac{e^{-j2\pi vy}}{-j2\pi v}\right]_{-Y/2}^{Y/2},$$

$$= \frac{1}{-j2\pi u}\left[e^{-juX} - e^{juX}\right]\frac{1}{-j2\pi v}\left[e^{-jvY} - e^{jvY}\right],$$

$$= XY\left[\frac{sin(\pi Xu)}{\pi Xu}\right]\left[\frac{sin(2\pi Yv)}{\pi Yv}\right]$$

$$= XY\mathrm{sinc}(\pi \mathrm{Xu})\mathrm{sinc}(\pi \mathrm{Yv}).$$

$f(x,y)$ $|F(u,v)|$



$|F(u,v)|$

이산 푸리에 변환

- 2D Discrete Fourier Transform

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-2\pi j(\frac{ux}{M}+\frac{vy}{N})}, \qquad u = 0,\cdots,M-1, \;\; v = 0,\cdots,N-1$$

$$= \; F_{real}(u,v) + \; j\, F_{imag}(u,v)$$

$$F_{real}(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\cos 2\pi(\frac{ux}{M}+\frac{vy}{N})$$

$$F_{imag}(u,v) = -\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\sin 2\pi(\frac{ux}{M}+\frac{vy}{N})$$

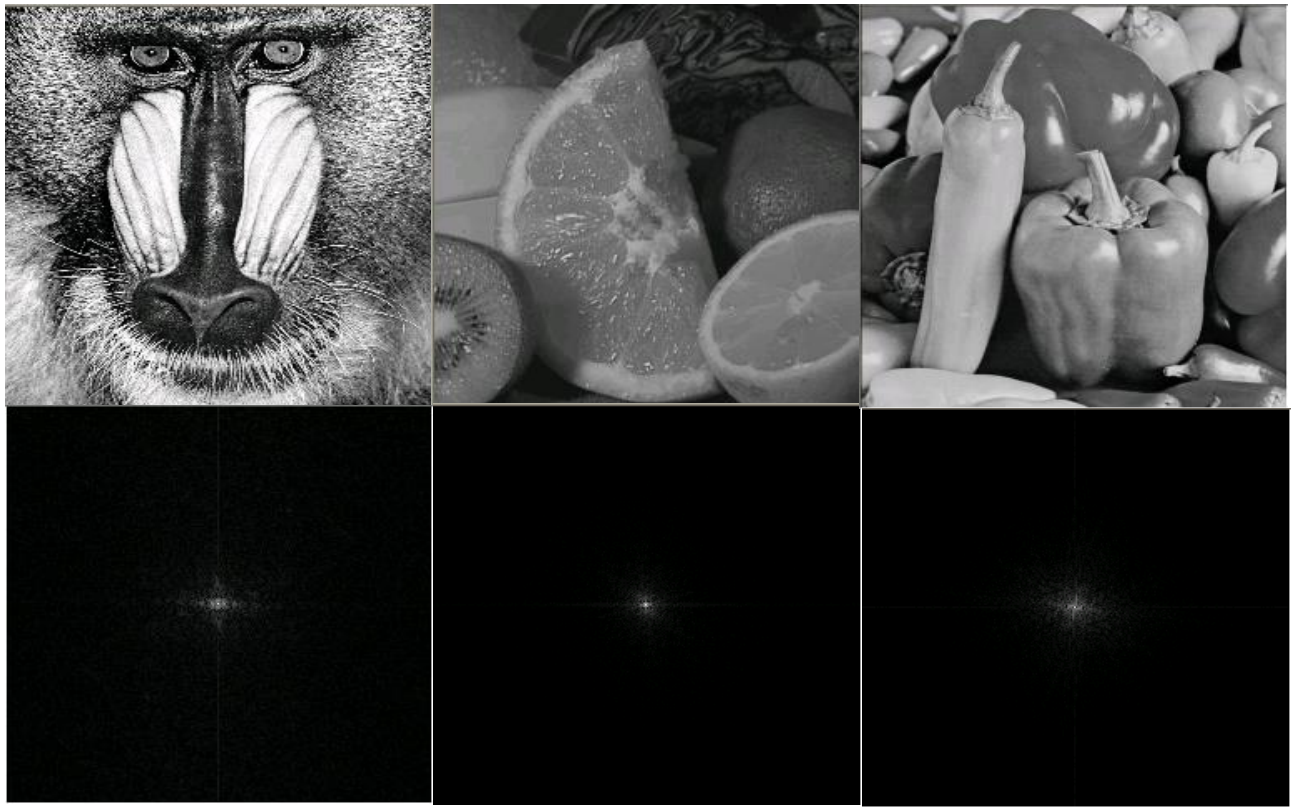
$F(0,0) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)$: average intensity of image

❖ FFT (Fast Fourier Transform)

M^2N^2 Multiplications $\rightarrow$ $MN\,log_2MN$ multiplications

이산 푸리에 변환

- 예



이산 푸리에 변환

- OpenCV 함수: dft()

```
void dft(InputArray src, OutputArray dst, int flags=0, int nonzeroRows=0)
```

매개 변수	설명
src	입력 행렬(실수이거나 복소수)
dst	출력 행렬(플래그에 따라서 크기와 유형이 달라진다)
flags	변환 플래그. 다음과 같은 값들의 조합이다. • DFT_INVERSE 역변환 • DFT_SCALE 결과값을 스케일한다. 결과를 행렬요수의 수로 나눈다. • DFT_COMPLEX_OUTPUT 수행결과가 복소수가 된다. • DFT_REAL_OUTPUT 수행결과가 실수가 된다.

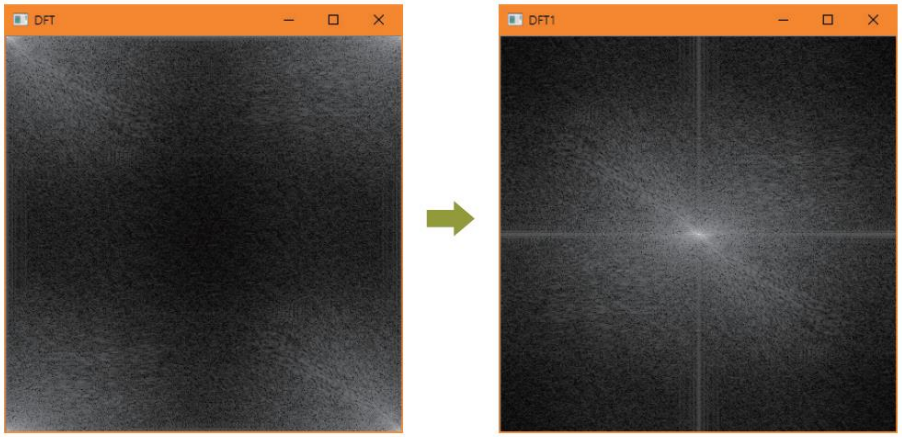
실습 1- DFT

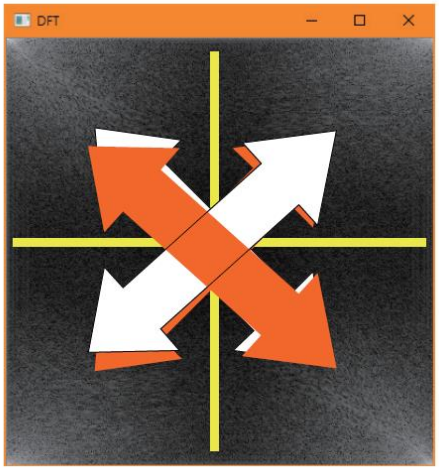
```
int main()
{
    Mat src = imread("d:/lenna.jpg", CV_LOAD_IMAGE_GRAYSCALE);
    Mat src_float;
    Mat dft_image;

    // 그레이스케일 영상을 실수 영상으로 변환한다.
    src.convertTo(src_float, CV_32FC1, 1.0 / 255.0);
    dft(src_float, dft_image, DFT_COMPLEX_OUTPUT);
    shuffleDFT(dft_image);
    displayDFT(dft_image);
    return 1;
}
```

```
void displayDFT(Mat& src)
{
    Mat image_array[2] = { Mat::zeros(src.size(), CV_32F), Mat::zeros(src.size(),CV_32F) };
    // ① DFT 결과 영상을 2개의 영상으로 분리한다.
    split(src, image_array);
    Mat mag_image;
    // ② 푸리에 변환 계수들의 절대값을 계산한다.
    magnitude(image_array[0], image_array[1], mag_image);

    // ③ 푸리에 변환 계수들은 상당히 크기 때문에 로그 스케일로 변환한다.
    // 0값이 나오지 않도록 1을 더해준다.
    mag_image += Scalar::all(1);
    log(mag_image, mag_image);

    // ④ 0에서 255로 범위로 정규화한다.
    normalize(mag_image, mag_image, 0, 1, CV_MINMAX);
    imshow("DFT", mag_image);
    waitKey(0);
}
```

```
void shuffleDFT(Mat& src)
{
    int cX = src.cols / 2;
    int cY = src.rows / 2;

    Mat q1(src, Rect(0, 0, cX, cY));
    Mat q2(src, Rect(cX, 0, cX, cY));
    Mat q3(src, Rect(0, cY, cX, cY));
    Mat q4(src, Rect(cX, cY, cX, cY));

    Mat tmp;
    q1.copyTo(tmp);
    q4.copyTo(q1);
    tmp.copyTo(q4);
    q2.copyTo(tmp);
    q3.copyTo(q2);
    tmp.copyTo(q3);
}
```

실습 1

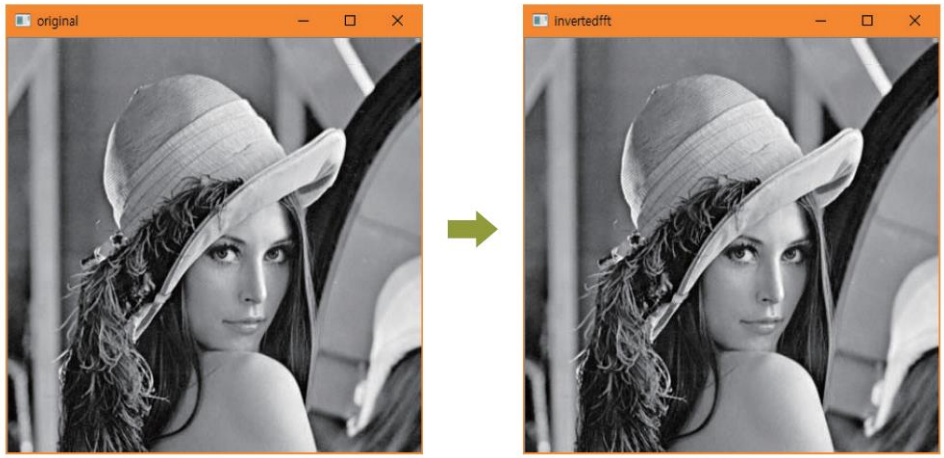




셔플링

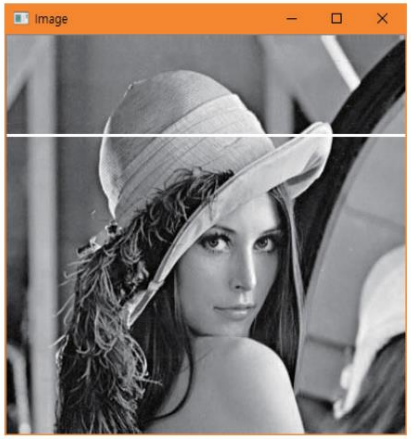
실습 2 - 역변환

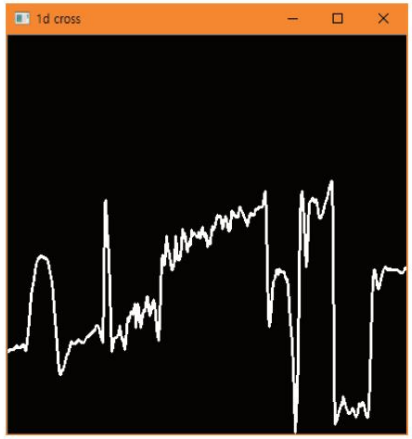
```
int main()
{
    Mat img = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);

    Mat img_float, dft1, inversedft, inversedft1;
    img.convertTo(img_float, CV_32F);
    dft(img_float, dft1, DFT_COMPLEX_OUTPUT);

    // 역변환을 수행한다.
    idft(dft1, inversedft, DFT_SCALE | DFT_REAL_OUTPUT);

    inversedft.convertTo(inversedft1, CV_8U);
    imshow("invertedfft", inversedft1);

    imshow("original", img);
    waitKey(0);
    return 0;
}
```

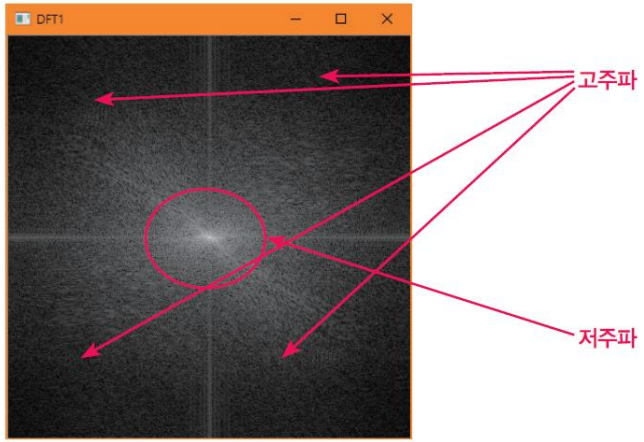


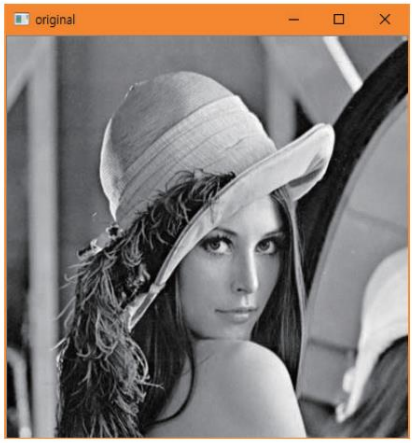
주파수 필터링

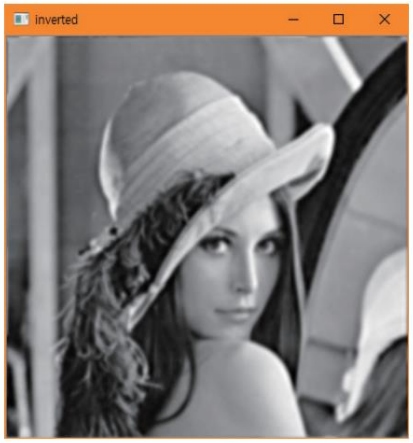
- 공간주파수 = 화소값들의 변화율





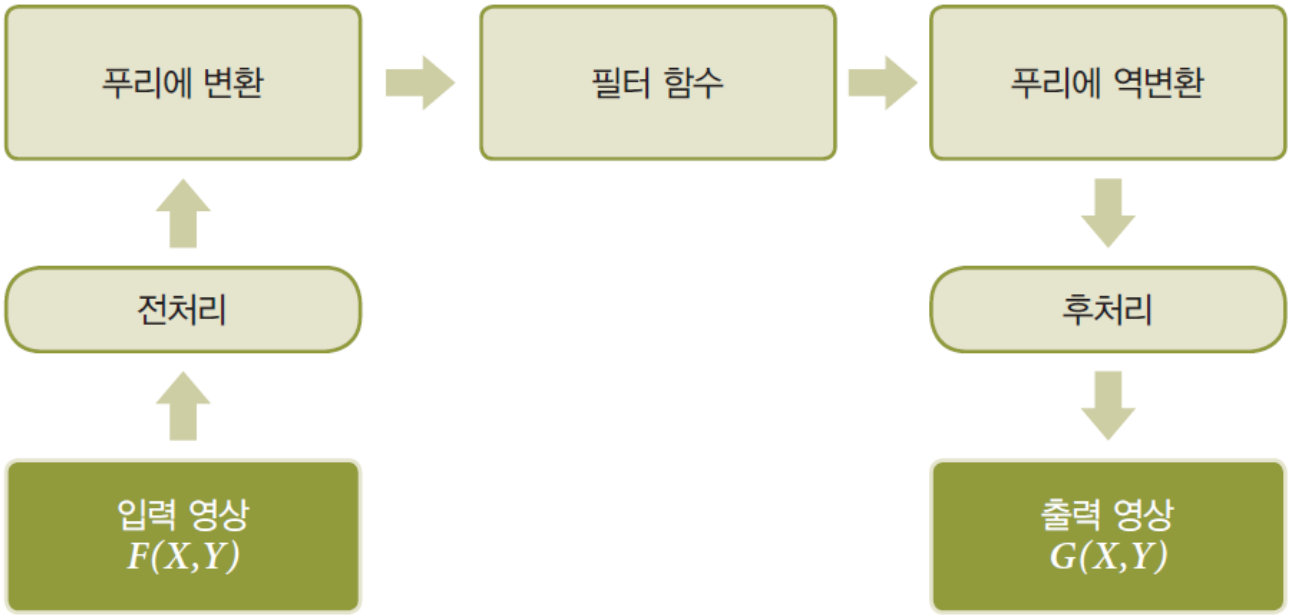






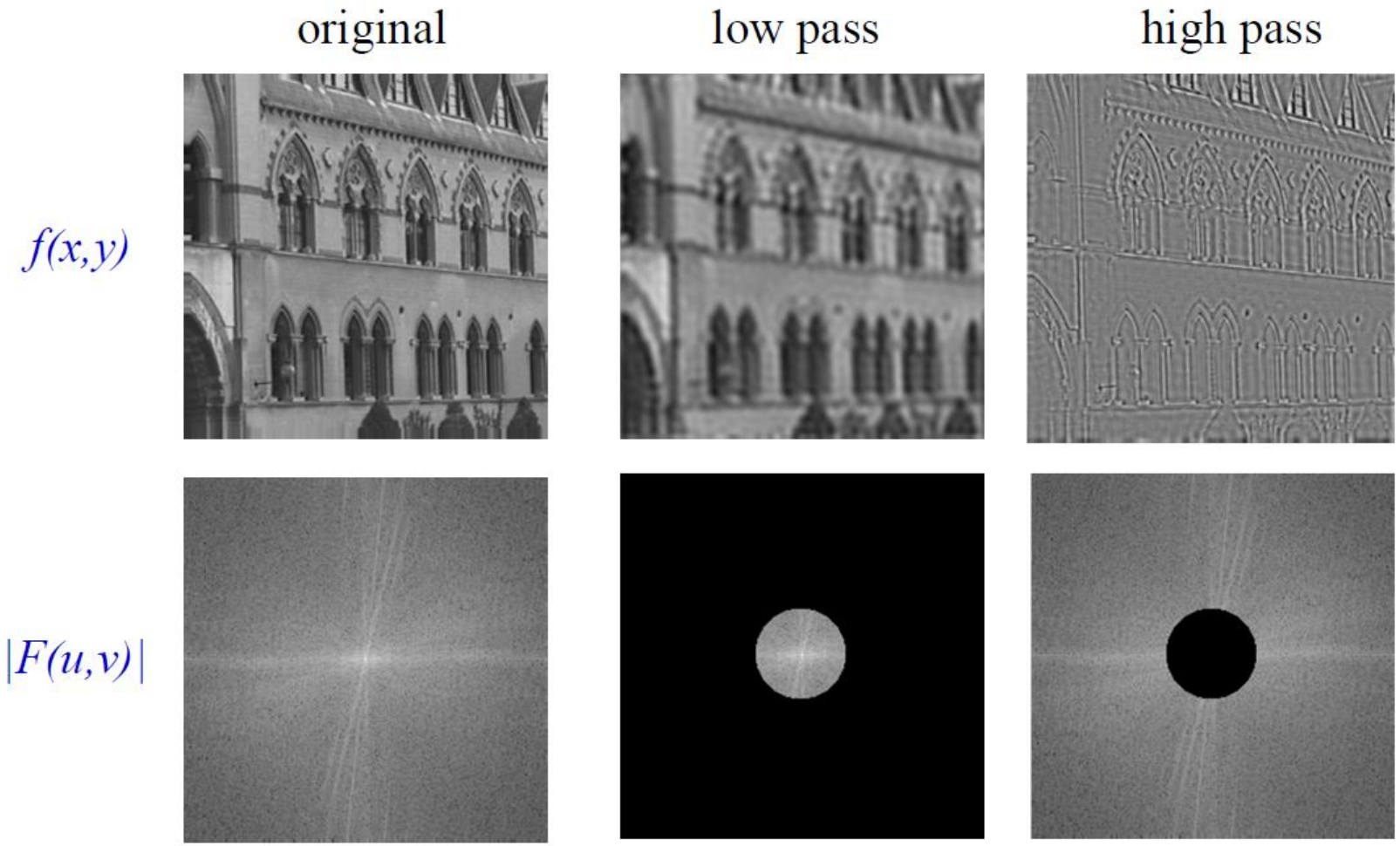
고주파 영상 저주파 영상

주파수 필터링



주파수 필터링

- Filtering

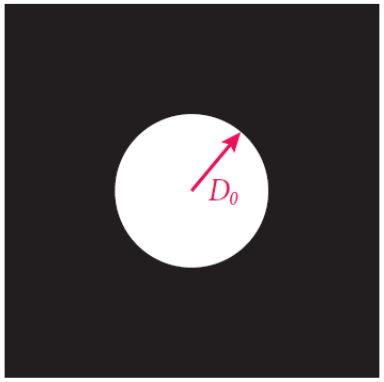


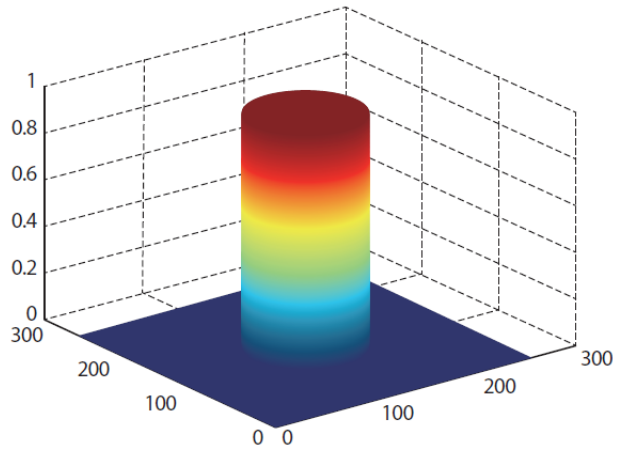
주파수 필터링

- 저주파 통과 필터

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

$$D(u, v) = \sqrt{u^2 + v^2}$$

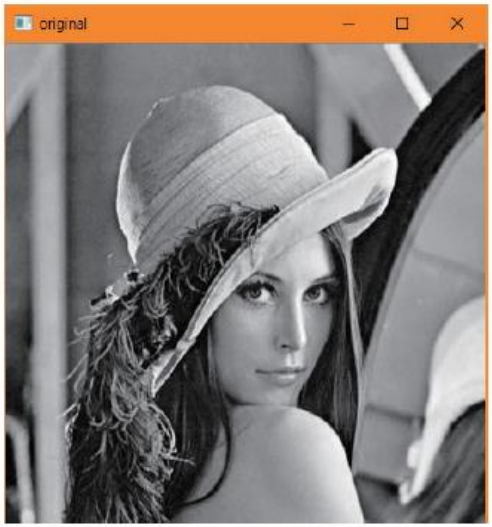


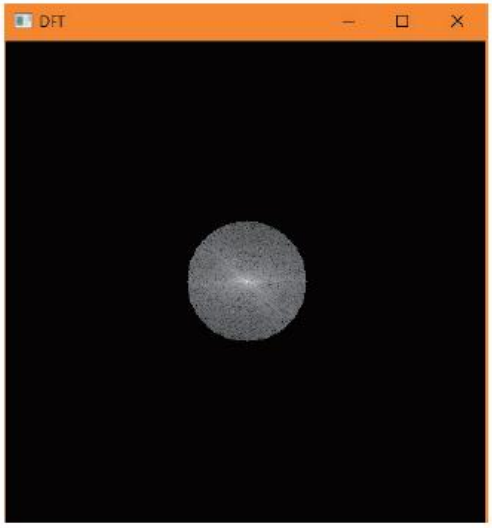


실습 3 - 저주파 통과 필터

```
void shuffleDFT(Mat& src) { ... }
void displayDFT(Mat& src) { ... }

// 원형 필터를 만든다.
Mat getFilter(Size size)
{
    Mat filter(size, CV_32FC2, Vec2f(0, 0));
    circle(filter, size / 2, 50, Vec2f(1, 1), -1);
    return filter;
}
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    Mat src_float;
    imshow("original", src);
    // 그레이스케일 영상을 실수 영상으로 변환한다.
    src.convertTo(src_float, CV_32FC1, 1.0 / 255.0);
    Mat dft_image;
    dft(src_float, dft_image, DFT_COMPLEX_OUTPUT);
    shuffleDFT(dft_image);

    Mat lowpass = getFilter(dft_image.size());
    Mat result;

    // 원형 필터와 DFT 영상을 서로 곱한다.
    multiply(dft_image, lowpass, result);
    displayDFT(result);

    Mat inverted_image;
    shuffleDFT(result);
    idft(result, inverted_image, DFT_SCALE | DFT_REAL_OUTPUT);
    imshow("inverted", inverted_image);
    waitKey(0);
    return 1;
}
```

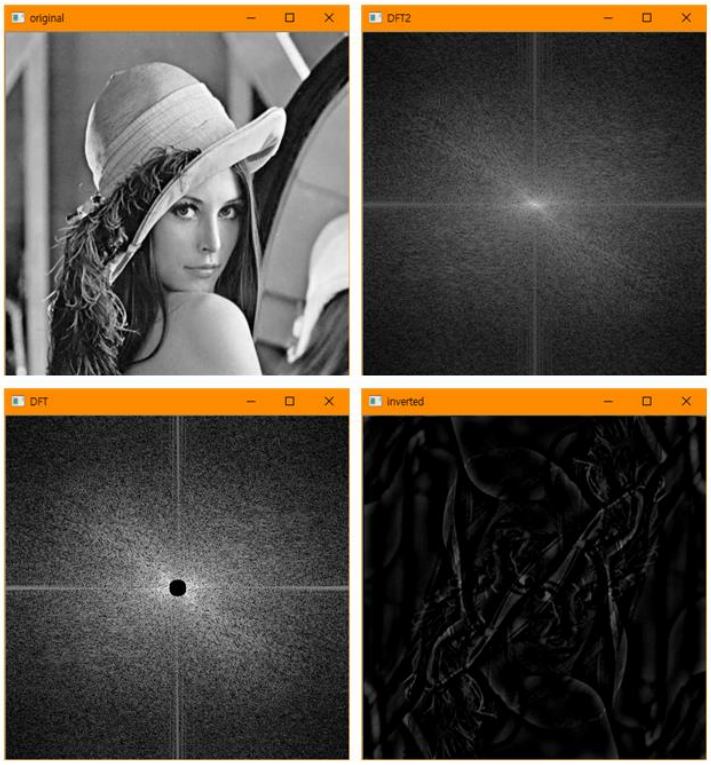
실습 3





실습 4 – 고주파 통과 필터

```
Mat getFilter(Size size)
{
    Mat filter = Mat::ones(size, CV_32FC2);
    circle(filter, size / 2, 10, Vec2f(0, 0), -1);
    return filter;
}
```

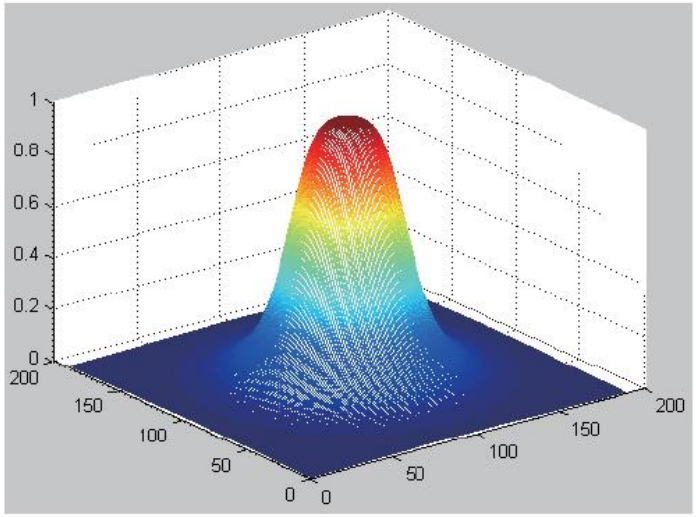


주파수 필터링

- Butter-worth 필터

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

$$D(u,v) = \sqrt{u^2+v^2}, \quad n = 1,2,...$$

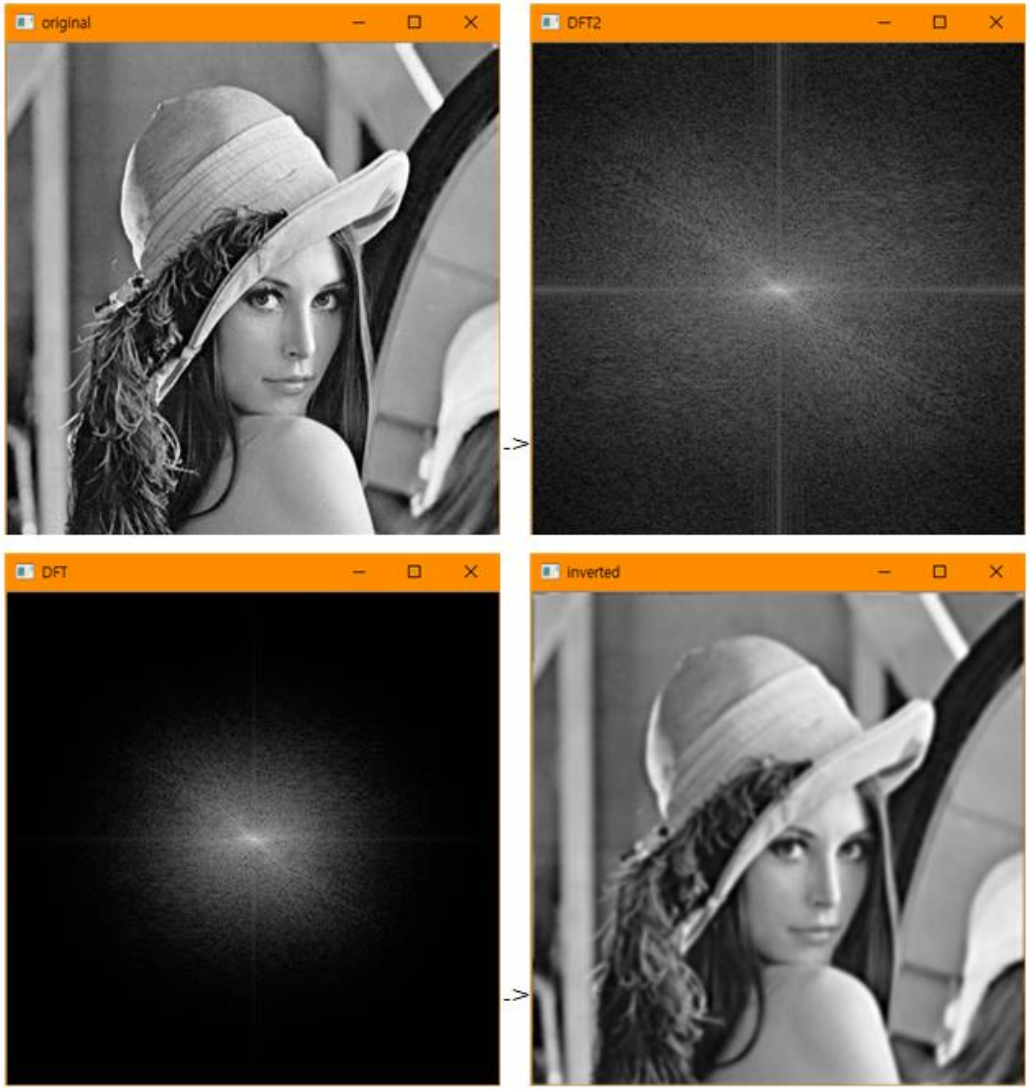


✓ 저주파 필터에 비해 자연스러운 영상 생성

실습 4 – butter-worth 필터

```
// 버터워쓰 필터를 만든다.
Mat getFilter(Size size)
{
    Mat tmp = Mat(size, CV_32F);
    Point center = Point(tmp.rows / 2, tmp.cols / 2);
    double radius;
    double D = 50;
    double n = 2;

    for (int i = 0; i < tmp.rows; i++) {
        for (int j = 0; j < tmp.cols; j++) {
                radius = (double)sqrt(pow((i - center.x), 2.0) + pow((double)(j - center.y), 2.0));
                tmp.at<float>(i, j) = (float)(1 / (1 + pow((double)(radius / D), (double)(2 * n))));
        }
    }
    Mat toMerge[] = { tmp, tmp };
    Mat filter;
    merge(toMerge, 2, filter);
    return filter;
}
```

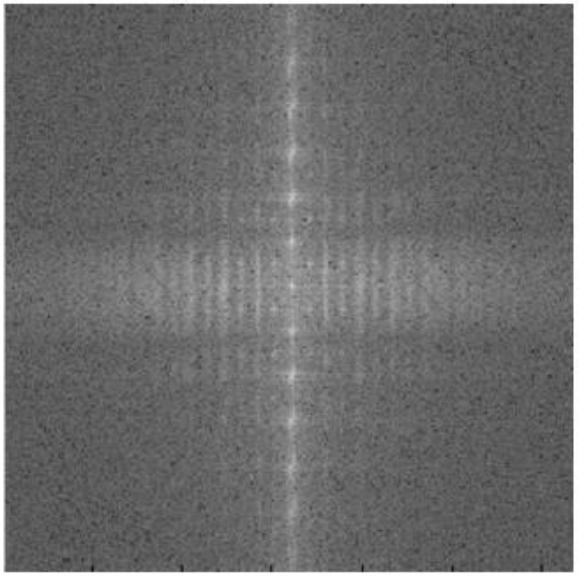
실습 4



주기적인 패턴 제거

- Periodic Structure

$f(x,y)$



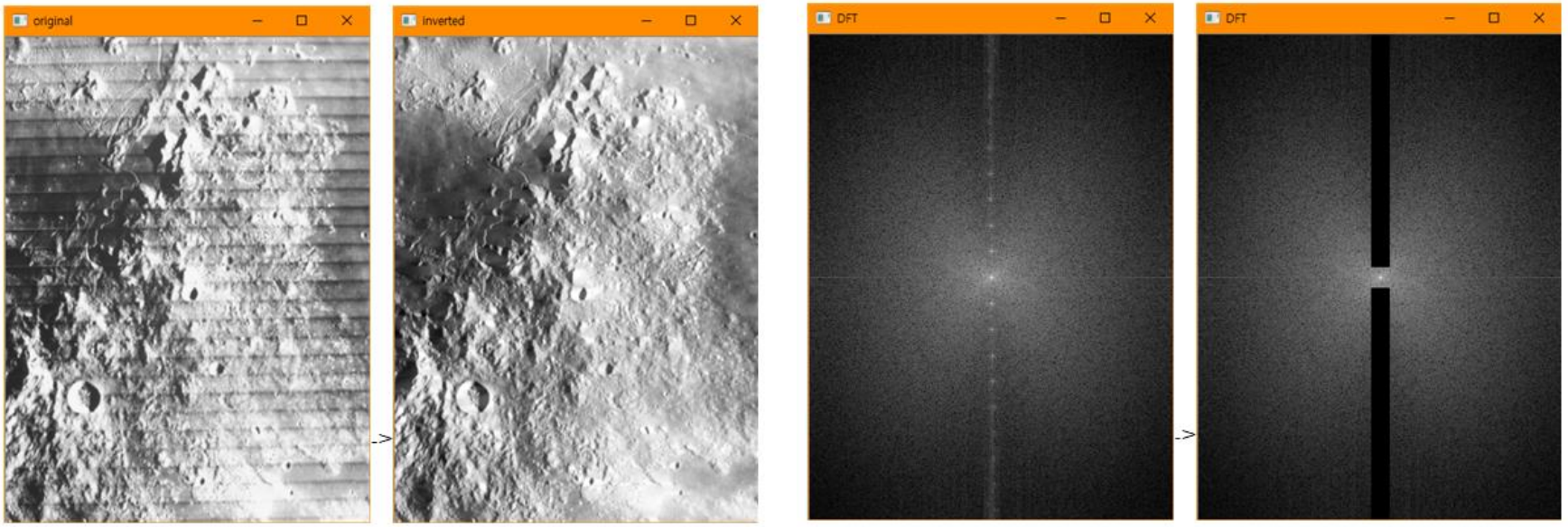
$|F(u,v)|$

FT has peaks at spatial frequencies of repeated texture

주기적인 패턴 제거

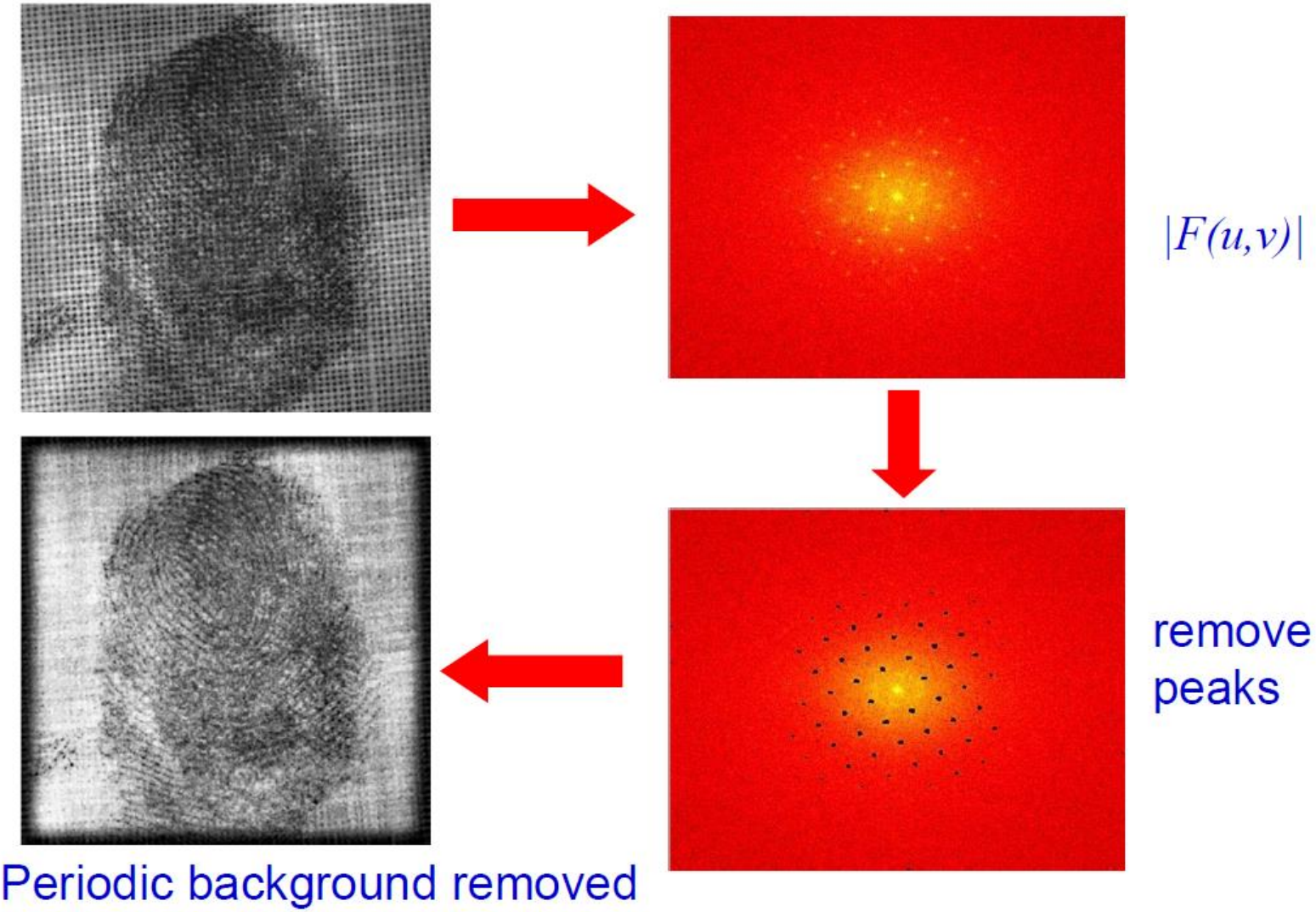
- 원본 영상의 훼손 없이 주기적인 패턴제거

  => 원점 부근 주파수만 남기고 수직 주파수 제거



주기적인 패턴제거

- Forensic application



실습 5 - 주기적인 패턴제거

```
// 필터를 만든다.
Mat getFilter(Size size)
{
    Mat tmp = Mat(size, CV_32F);

    for (int i = 0; i < tmp.rows; i++) {
        for (int j = 0; j < tmp.cols; j++) {
            if (j>(tmp.cols/2 - 10) && j<(tmp.cols/2 + 10) && i >(tmp.rows/2 + 10)) tmp.at<float>(i, j) = 0;
            else if (j>(tmp.cols/2 - 10) && j<(tmp.cols/2 + 10) && i <(tmp.rows/2 - 10)) tmp.at<float>(i, j) = 0;
            else tmp.at<float>(i, j) = 1;
      }
    }
    Mat toMerge[] = { tmp, tmp };
    Mat filter;
    merge(toMerge, 2, filter);
    return filter;
}
```

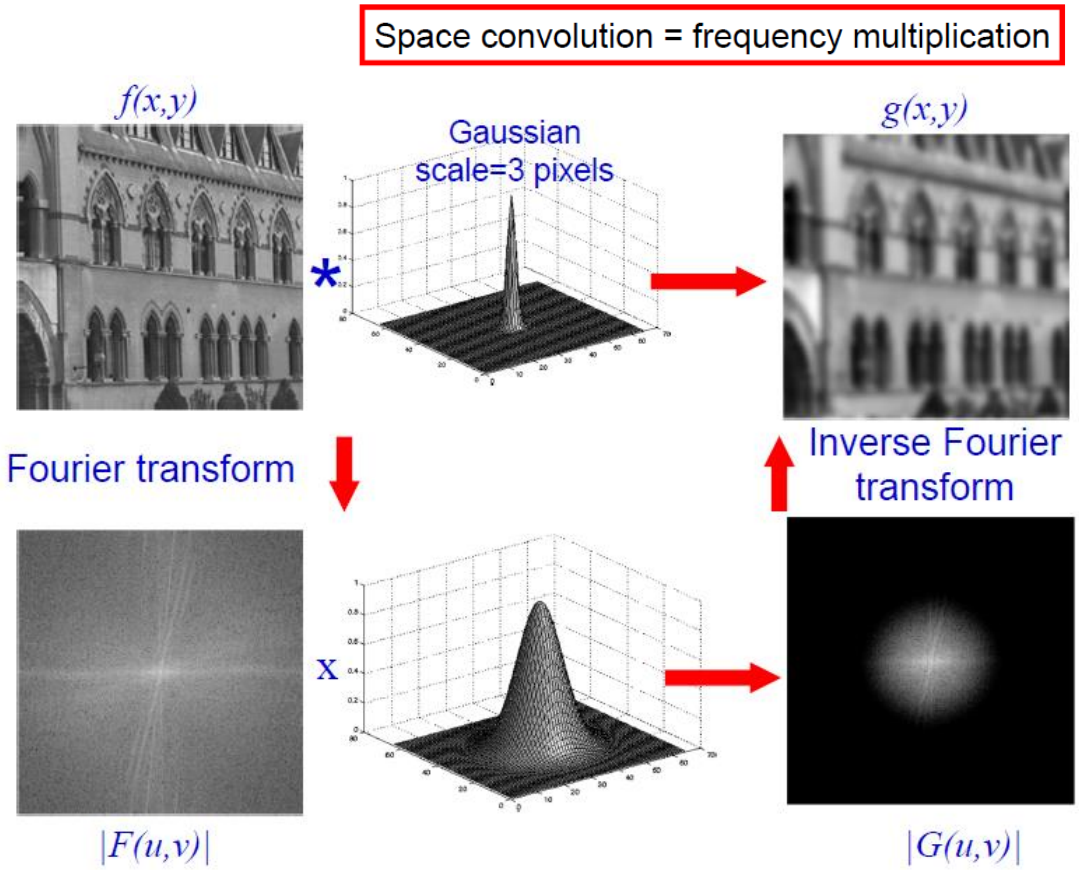
실습 5 - 주기적인 패턴제거

```
int main()
{
    Mat src = imread("d:/lunar.png", IMREAD_GRAYSCALE);
    Mat src_float, dft_image;
    imshow("original", src);

    // 그레이스케일 영상을 실수 영상으로 변환한다.
    src.convertTo(src_float, CV_32FC1, 1.0 / 255.0);
    dft(src_float, dft_image, DFT_COMPLEX_OUTPUT);
    shuffleDFT(dft_image);
    displayDFT(dft_image);

    Mat lowpass = getFilter(dft_image.size());
    Mat result;
    // 필터와 DFT 영상을 서로 곱한다.
    multiply(dft_image, lowpass, result);
    displayDFT(result);

    Mat inverted_image;
    shuffleDFT(result);
    idft(result, inverted_image, DFT_SCALE | DFT_REAL_OUTPUT);
    imshow("inverted", inverted_image);
    waitKey(0);
    return 1;
}
```

주파수 영상처리 응용

- Convolution (mask operation)

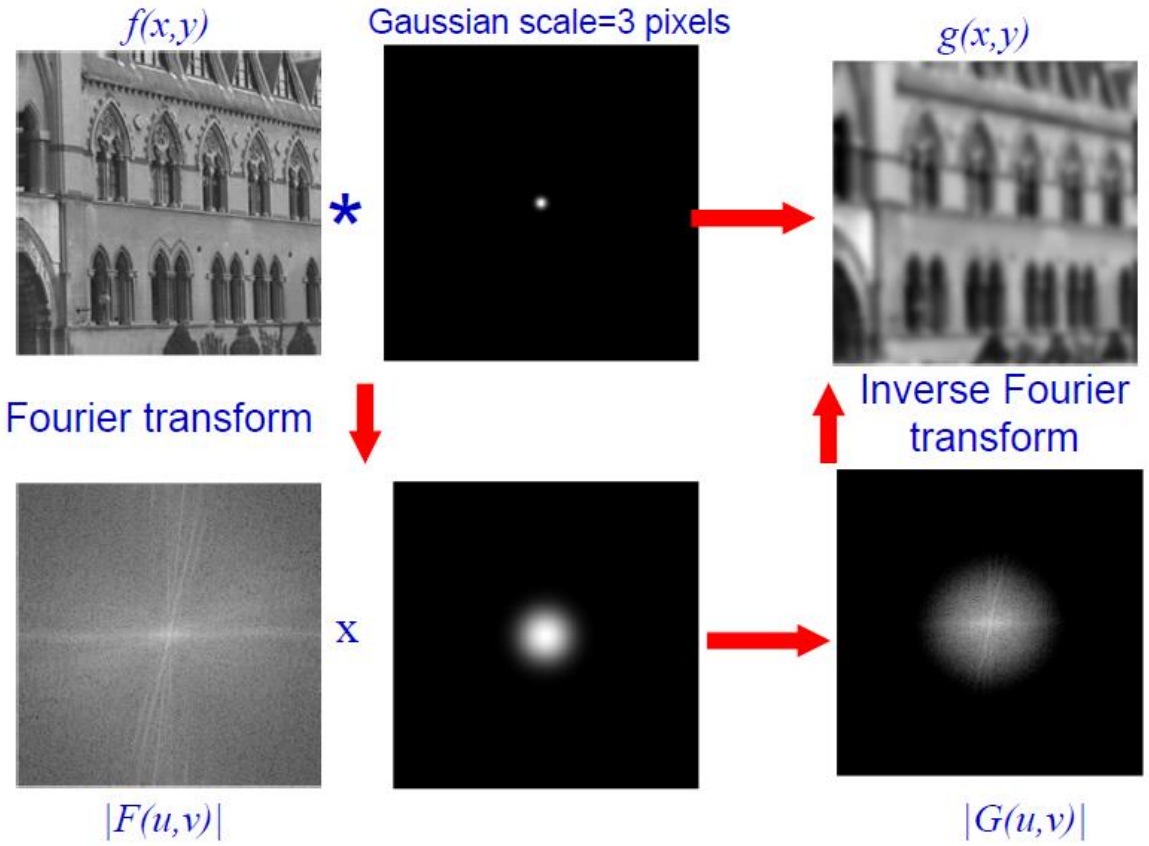
$$f(x,y) * h(x,y) \quad \Leftrightarrow \quad F(u,v)H(u,v)$$



1. Compute FT of image and FT of Gaussian
2. Multiply FT's
3. Compute inverse FT of the result.

주파수 영상처리 응용

- Convolution



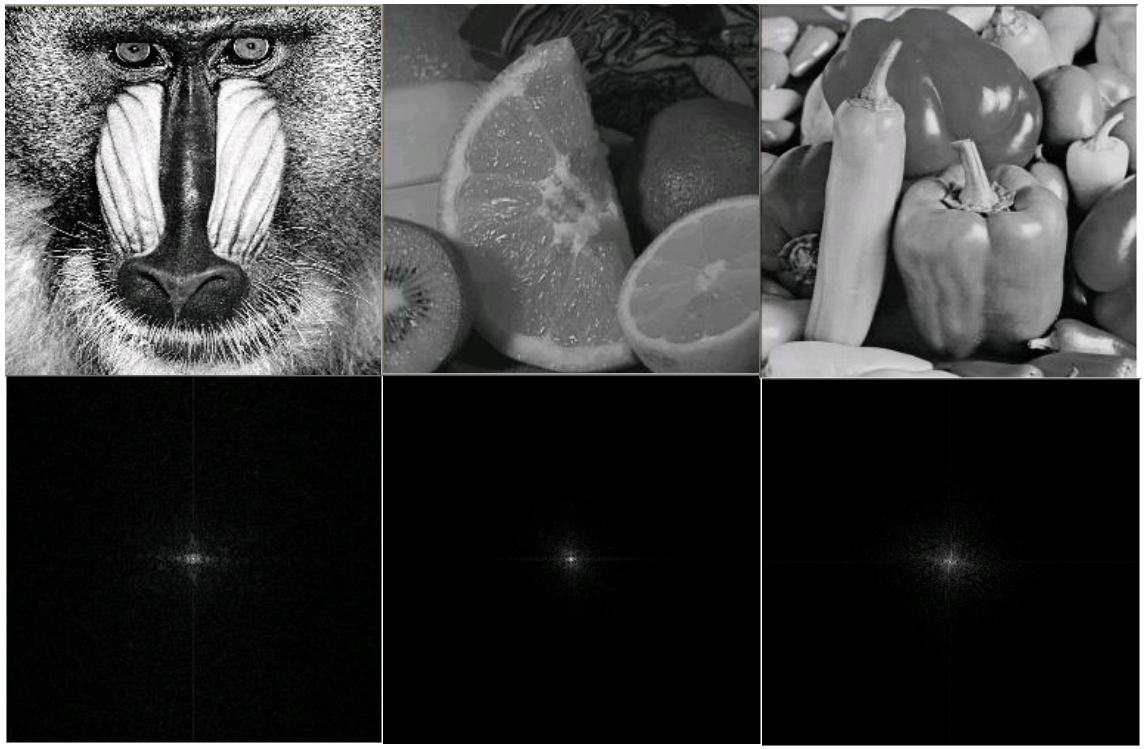
➢ Frequency domain 에서의 연산처리가 빠른 경우에 적용 (예: 마스크 크기가 큰 경우)

주파수 영상처리 응용

- Image Compression (영상 압축)
  (ex) cosine transformation: JPEG, MPEG
  (ex) wavelet transformation: MPEG4

original

compressed



HW

1. 교재 Exercise 3번
2. 교재 Exercise 4번