Chapter

9

# 변환영역처리

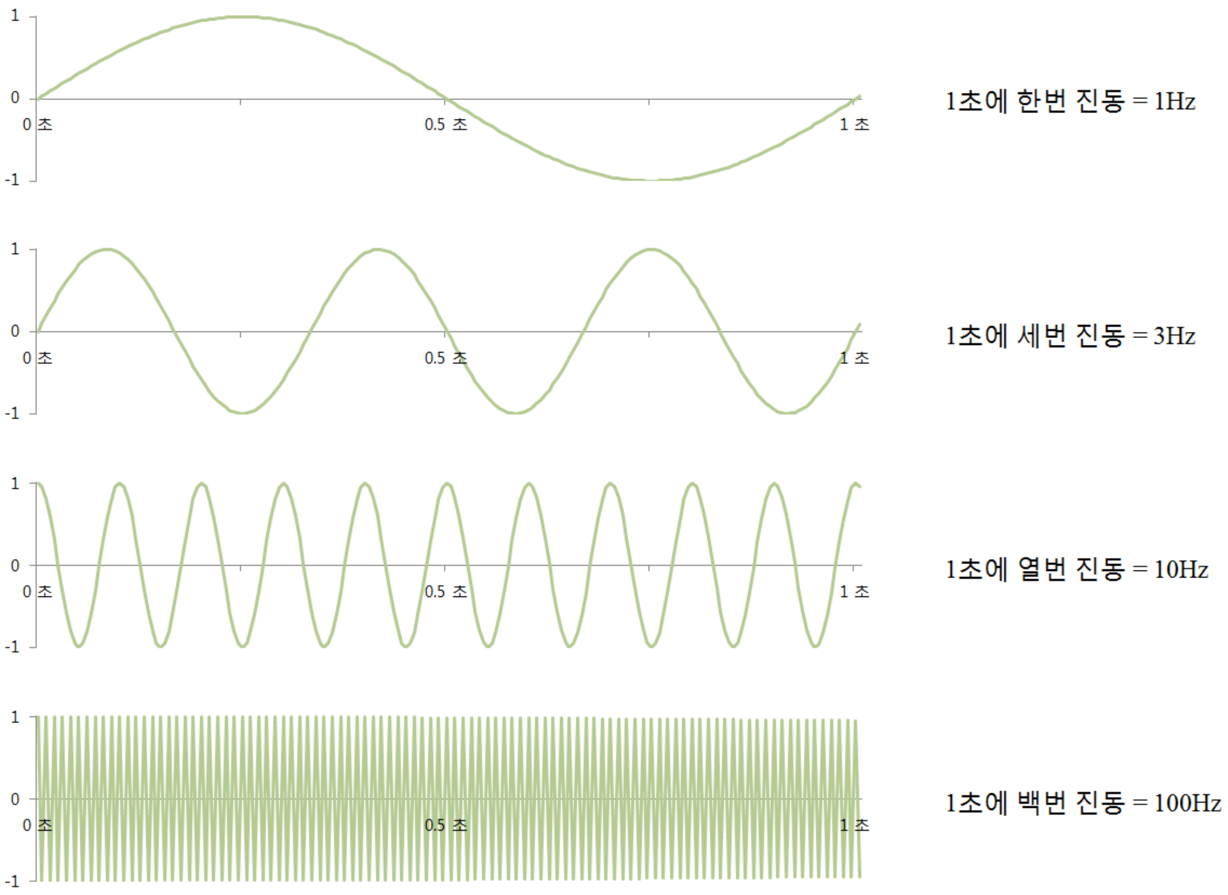
1

# 변환영역처리의 필요성

# 공간 주파수

## ❖ 주파수(frequency)

- 어떤 신호가 주기적으로 발생하는 빈도
- 주파수 단위(Hz)
  - 1초 동안에 진동하는 횟수




## ❖ 공간 주파수

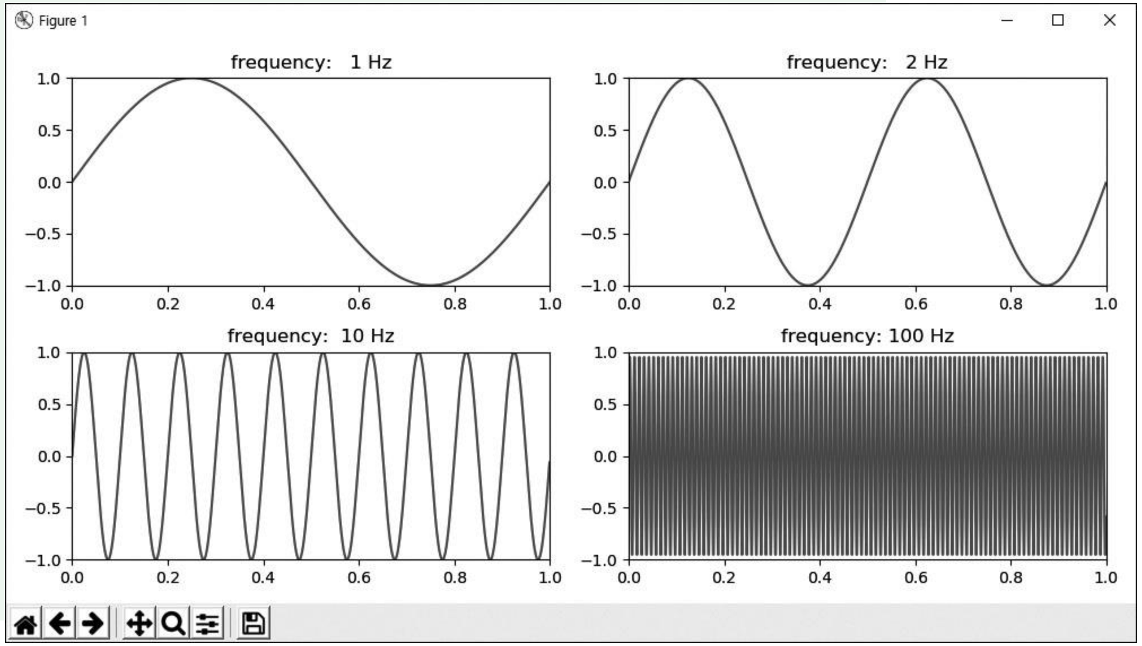
- Spatial frequency
- 영상에서

  밝기의 변화가 주기적으로

  발생하는 빈도

## 예제 9.1.1 주파수 그리기 |– 01.frequence.py

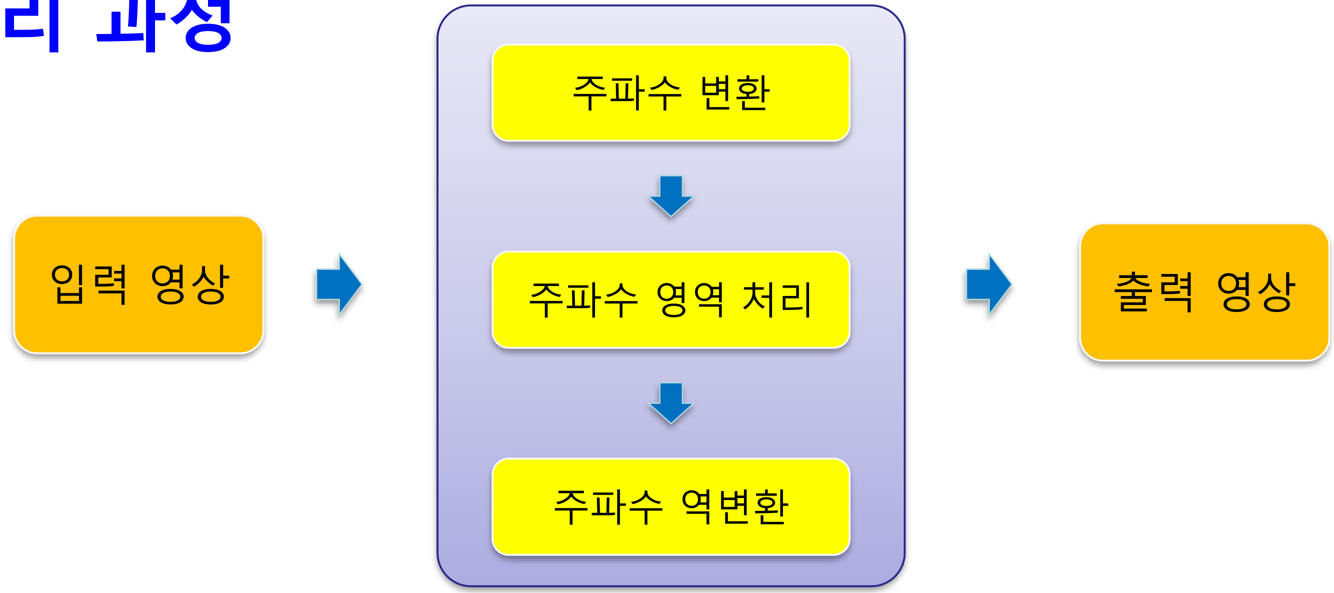
```
import matplotlib.pyplot as plt                # 그래프그리기 라이브러리 임포트
import numpy as np

t = np.arange(0, 1, 0.001)                     # 샘플링 범위 및 개수
Hz = [1, 2, 10, 100]                           # 주파수 예시
gs = [np.sin(2 * np.pi * t * h) for h in Hz]   # sin 함수 계산

plt.figure(figsize=(10, 5))
for i, g in enumerate(gs):
        plt.subplot(2, 2, i+1), plt.plot(t, g)
        plt.xlim(0, 1), plt.ylim(-1, 1 )
        plt.title("frequency: %3d Hz" % Hz[i])
plt.tight_layout()
plt.show()
```



# 공간 주파수

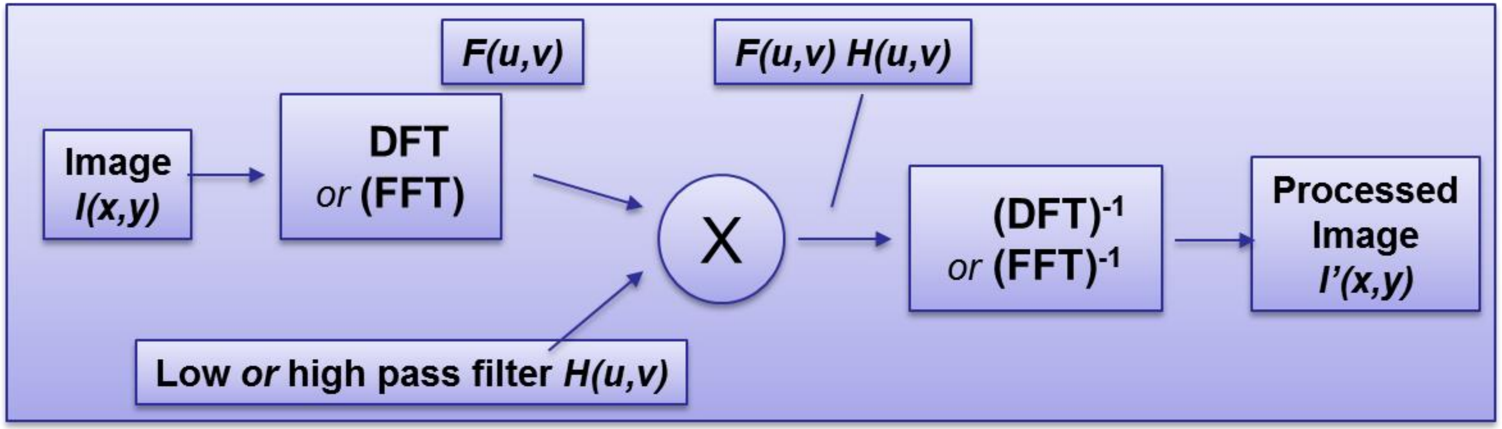
- **밝기의 변화 정도에 따라서 고주파 영역/ 저주파 영역으로 분류**
- **저주파 공간 영역**
  - 화소 밝기가 거의 변화가 없거나 점진적으로 변화하는 것
  - 영상에서 배경 부분이나 객체의 내부에 많이 있음
- **고주파 공간 영역**
  - 화소 밝기가 급변하는 것
  - 경계부분이나
    객체의 모서리 부분

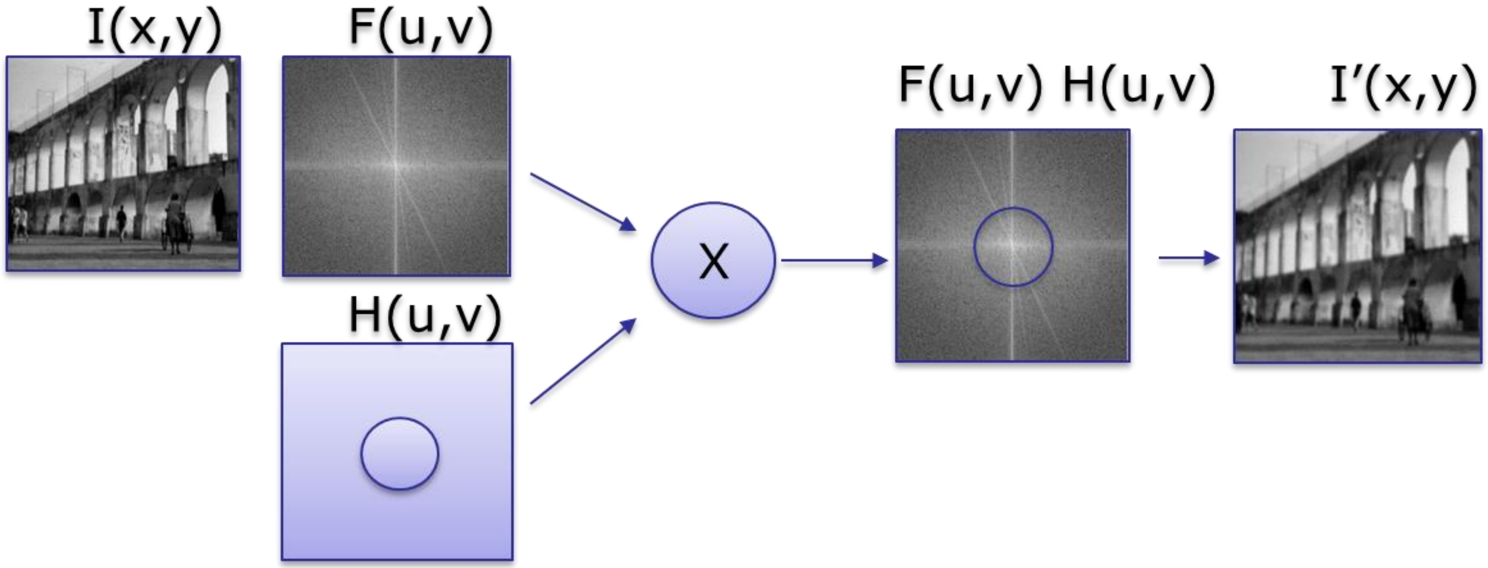
- **영상을 주파수 영역별로 분리 한다면 ?**
  - 경계부분에 많은 고주파 성분 제거한 영상 → 경계 흐려진 영상
  - 고주파 성분만 취한 영상 → 경계나 모서리만 포함하는 영상(에지 영상)
- **공간 영역상에서 저주파 성분과 고주파 성분이 혼합 → 영역 분리해서 선별적 처리 어려움 → 변환영역 처리 필요**
- **변환 영역 처리 과정**



# 영상처리 방법의 예

❖ **low & high pass filtering**



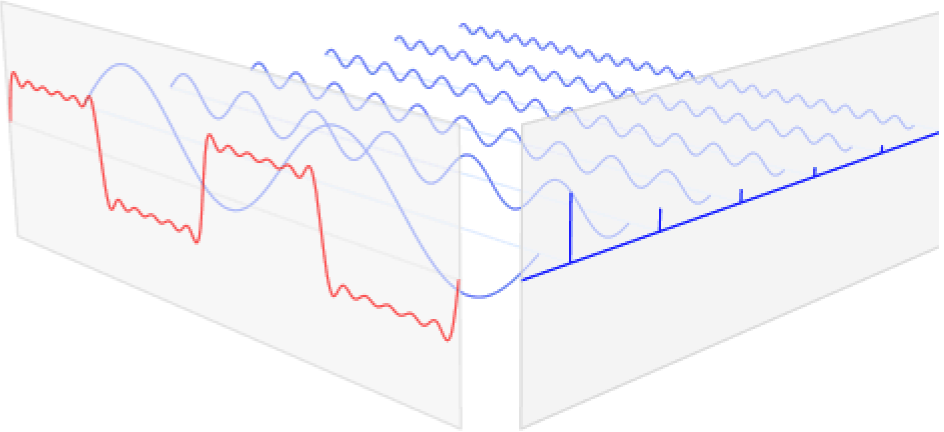
2

# Discrete Fourier Transform

# 푸리에 변환(Fourier Transform)

## ❖ 푸리에 변환(Fourier Transform)

- 신호 또는 영상을 **주파수영역으로 변환**하는 가장 일반적인 방법
- 입력 신호를 **다양한 주파수를 갖는 기저함수들의 합으로 분해**하여 표현하는 방법
  - 모든 신호는 주기함수(sin 또는 cos함수)들의 합으로 표현됨 → 따라서 모든 신호는 주기함수(기저함수)로 분해될 수 있음



푸리에 변환 (그림출처: 위키피디아)

$$g(t) = \int_{-\infty}^{\infty} G(f) e^{j2\pi ft} df$$

$$G(f) = \int_{-\infty}^{\infty} g(t) e^{-j2\pi ft} dt$$

# 신호의 예

❖ 여러 개의 sin함수로 어떤 신호들을 생성할 수 있음
→ 반대로 어떤 신호는 여러 개의 sin함수로 분리될 수 있음

$$g(t) = 0.3 * g_1(t) - 0.7 * g_2(t) + 0.5 * g_3(t)$$

$$g_1(t) = \sin(2\pi * t)$$

$$g_2(t) = \sin(2\pi * 3t)$$

$$g_3(t) = \sin(2\pi * 5t)$$

❖ 즉, 계수 $G(f)$를 가진 기저함수들 $g_f(t)$의 합으로 표현할 수 있음

$$g(t) = \int_{-\infty}^{\infty} G(f) \cdot g_f(t) df$$

# Fourier Transform

- ❖ **푸리에 변환에서 사용하는 기저함수**

$$g_f(t) = \cos(2\pi ft) + j \cdot \sin(2\pi ft) = e^{j2\pi ft}$$

오일러 공식(Euler's formula)

$$e^{j\theta} = cos\theta + j \cdot sin\theta$$

- ❖ **1차원 연속 푸리에 변환**

$$g(t) = \int_{-\infty}^{\infty} G(f)e^{j2\pi ft}df$$

- ❖ **1차원 연속 역푸리에 변환**

$$G(f) = \int_{-\infty}^{\infty} g(t)e^{-j2\pi ft}dt$$

$$j = \sqrt{-1}$$

# 이산 푸리에 변환(DFT)

## ❖ DFT : Discrete Fourier Transform

- 디지털 신호(이산신호)에 적용

$$G(k) = \sum_{n=0}^{N-1} g[n] \cdot e^{-j2\pi k\frac{n}{N}}, \qquad k = 0, \dots, N-1$$

$$g[n] = \frac{1}{N}\sum_{n=0}^{N-1} G(k) \cdot e^{j2\pi k\frac{n}{N}}, \qquad n = 0, \dots, N-1$$

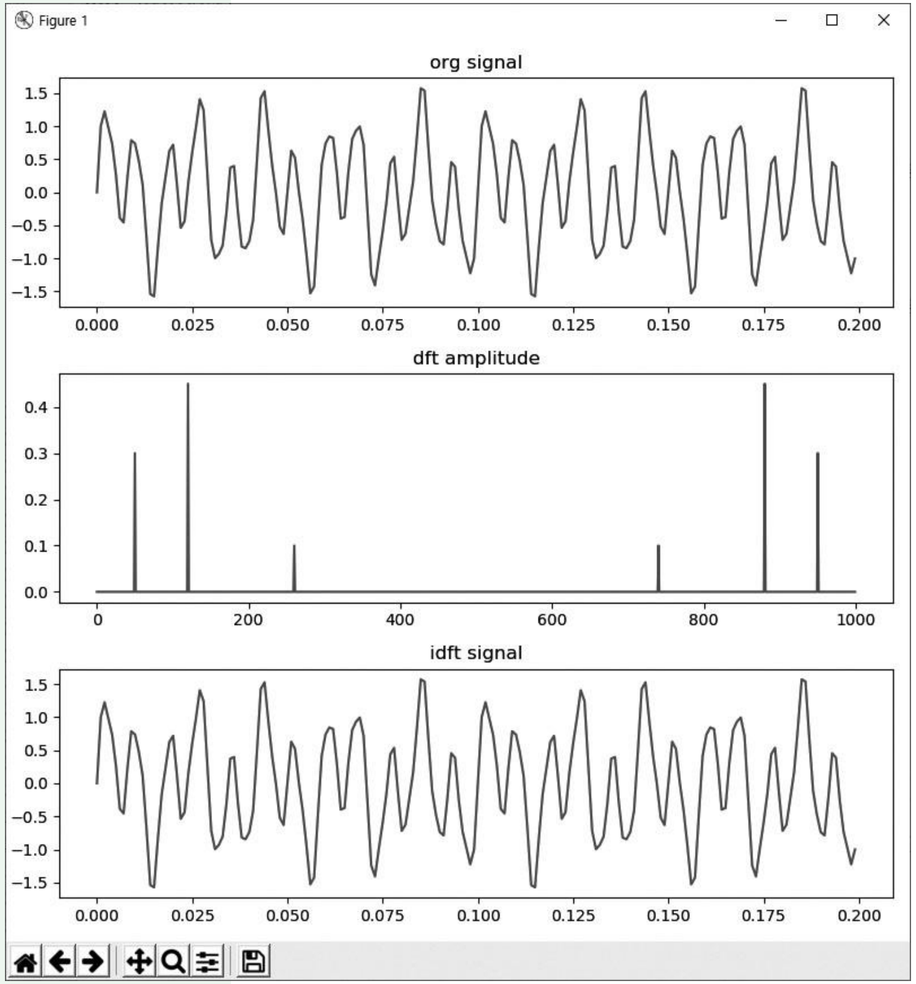
- $g[n]$ : 디지털 신호
- $G(k)$ : 주파수 k에 대한 푸리에 변환 계수
- k, n : 신호의 원소 개수

continue…

예제 9.2.2 1차원 이산 푸리에 변환 – 03.1d_dft.cpp

```
import numpy as np, math
import matplotlib.pyplot as plt

def exp(knN):
        th = -2 * math.pi * knN                          # 푸리에 변환 각도값
        return complex(math.cos(th), math.sin(th))       # 복소수 클래스

def dft(g):
        N = len(g)
        dst = [sum(g[n] * exp(k*n/N ) for n in range(N)) for k in range(N) ]
        return np.array(dst)

def idft(g):
        N = len(g)
        dst = [sum(g[n] * exp(-k*n/N) for n in range(N)) for k in range(N) ]
        return np.array(dst) / N

```

```
fmax = 1000                              # 샘플링 주파수 1000Hz: 최대주파수의 2배
dt = 1/fmax                              # 샘플링 간격
t = np.arange(0, 1, dt)                  # Time vector

g1 = np.sin(2 * np.pi * 50 * t)
g2 = np.sin(2 * np.pi * 120 * t)
g3 = np.sin(2 * np.pi * 260 * t)
g = g1 * 0.6 + g2 * 0.9 + g3 * 0.2      # 신호 합성

N = len(g)                               # 신호 길이
df = fmax/N                              # 샘플링 간격
f = np.arange(0, N, df)
xf = dft(g) * dt                         # 저자구현 푸리에 변환 함수
g2 = idft(xf) / dt

plt.figure(figsize=(10,10))
plt.subplot(3,1,1), plt.plot(t[0:200], g[0:200]), plt.title(“org signal”)
plt.subplot(3,1,2), plt.plot(f, np.abs(xf) ), plt.title(“dft amplitude”)
plt.subplot(3,1,3), plt.plot(t[0:200], g2[0:200]), plt.title(“idft signal”)
plt.show()
```



# 스펙트럼(spectrum)과 위상(phase)

## ❖ 1-d DFT의 경우

- 기저함수를 sin과 cos함수로 변경하여 표현

$$G(k) = \sum_{n=0}^{N-1} g[n] \cdot e^{-j2\pi k\frac{n}{N}}, \qquad k = 0, \dots, N-1$$

$$G(k) = \sum_{n=0}^{N-1} g[n] \cdot \left(\cos\left(-2\pi k\frac{n}{N}\right) + j \cdot \sin(-2\pi k\frac{n}{N})\right), \qquad k = 0, \dots, N-1$$

오일러 공식(Euler's formula)

$$e^{j\theta} = cos\theta + j \cdot sin\theta$$

## ❖ 실수부와 허수부를 구분하여 표현 → 복소수 행렬 생성됨

$$G(k)_{Re}=\sum_{n=0}^{N-1} g[n]_{Re}\cdot\cos\left(-2\pi k\frac{n}{N}\right)-g[n]_{Im}\cdot\sin\left(-2\pi k\frac{n}{N}\right)$$

$$G(k)_{Im}=\sum_{n=0}^{N-1} g[n]_{Im}\cdot\cos\left(-2\pi k\frac{n}{N}\right)+g[n]_{Re}\cdot\sin\left(-2\pi k\frac{n}{N}\right)$$

$$g[n]_{Re}=\frac{1}{N}\sum_{n=0}^{N-1} G(k)_{Re}\cdot\cos\left(2\pi k\frac{n}{N}\right)-G(k)_{Im}\cdot\sin\left(2\pi k\frac{n}{N}\right)$$

$$g[n]_{Im}=\frac{1}{N}\sum_{k=0}^{N-1} G(k)_{Im}\cdot\cos\left(2\pi k\frac{n}{N}\right)+G(k)_{Re}\cdot\sin\left(2\pi k\frac{n}{N}\right)$$

## ❖ 복소수의 행렬을 확인하려면 → 영상으로 표현

- 복소수의 실수부와 허수부를 벡터로 간주하여 벡터의 크기 계산

  → 주파수 **스펙트럼(spectrum)**
- 실수부와 허수부 원소의 기울기 계산

  → 주파수 **위상(phase)**

$$|G(k,l)| = \sqrt{Re(k,l)^2 + Im(k,l)^2}$$

$$|\theta(k,l)| = \tan^{-1}\left(\frac{Im(k,l)}{Re(k,l)}\right)$$

# 2차원 이산 푸리에 변환

## ❖ 2-d DFT

$$G(k,l) = \sum_{m=0}^{M-1}\left(\sum_{n=0}^{N-1} g[n,m]\cdot e^{-j2\pi k\frac{n}{N}}\right)\cdot e^{-j2\pi k\frac{m}{M}}, \qquad k=0,\ldots,N-1 \qquad l=0,\ldots,M-1$$

$$= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} g[n,m]\cdot e^{-j2\pi k\left(\frac{kn}{N}+\frac{lm}{M}\right)}$$

## ❖ 2-d Inverse DFT

$$g[n,m] = \frac{1}{NM}\sum_{m=0}^{M-1}\left(\sum_{n=0}^{N-1} G(k,l)\cdot e^{-j2\pi l\frac{n}{N}}\right)\cdot e^{-j2\pi l\frac{m}{M}}, \qquad k=0,\ldots,N-1 \qquad l=0,\ldots,M-1$$

$$= \frac{1}{NM}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} G(k,l)\cdot e^{j2\pi\left(\frac{kn}{N}+\frac{lm}{M}\right)}$$

# 푸리에 스펙트럼(Fourier spectrum)

❖ **해당 주파수 성분이 원 신호(이미지)에 얼마나 강하게 포함되어 있는지를 나타냄**

- 너비가 M이고, 높이가 N인 이미지를 푸리에 변환 → M*N의 $g[n,m]$을 얻을 수 있음
- 스펙트럼 $|g[n,m]|$을 픽셀값으로 사용하면 **푸리에 스펙트럼을 원본 이미지와 동일한 크기로 시각화** 할 수 있음

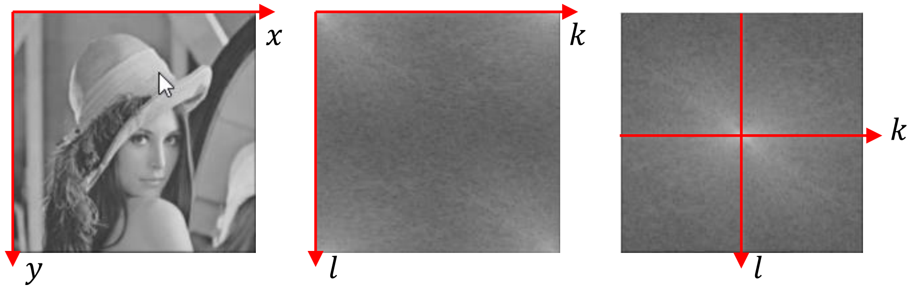
$$G(k,l) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} g[n,m] \cdot e^{-j2\pi k\left(\frac{kn}{N}+\frac{lm}{M}\right)}$$

$$g[n,m] = \frac{1}{NM}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} G(k,l) \cdot e^{j2\pi\left(\frac{kn}{N}+\frac{lm}{M}\right)}$$

## ❖ 주파수 스펙트럼 영상의 문제점

- 저주파 영역의 계수값이 고주파 영역에 비해 상대적으로 너무 큼
  - 계수값을 정규화해서 영상으로 표현하면 최저주파 영역만 흰색으로 나타나고 나머지 영역은 거의 검은색으로 나타남

    → 계수값에 **log함수 적용**

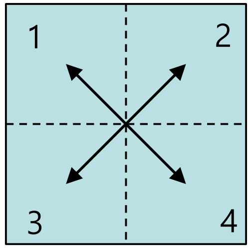



- 모서리로 갈 수록 값이 커지기 때문에 스펙트럼의 형태를 파악하기 힘듦
  - 저주파 영역이 모서리 부분에 위치, 고주파 부분이 중심부에 위치
  - 사각형의 각 모서리를 중심으로 원형의 밴드를 형성하여 주파수 영역이 분포
  - 해당 주파수 영역 처리시 불편함

    → 모양 변경 필요 → **셔플링(shuffling)**을 수행

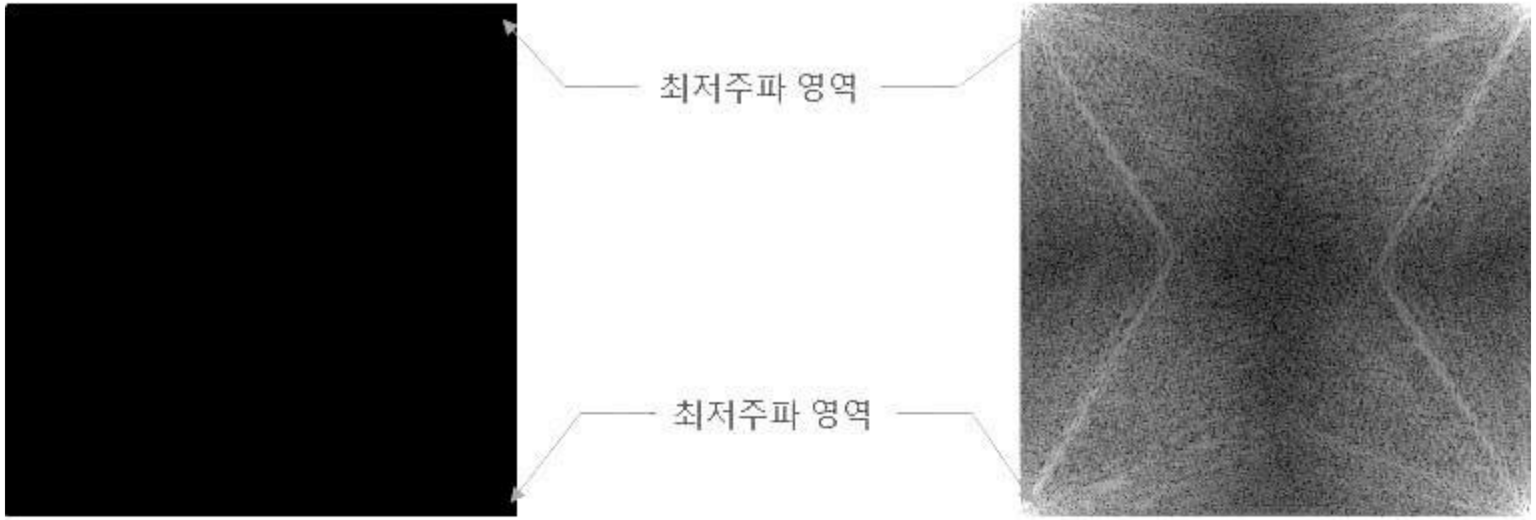
    » 1사분면과 3사분면의 영상 맞교환, 2사분면과 4사분면의 영상 맞교환

    » **시프트(shift)**라고도 함



## ❖ 로그함수 적용

- 저주파 영역의 계수값이 고주파 영역에 비해 상대적으로 너무 큼
- 계수값을 정규화해서 영상으로 표현하면 최저주파 영역만 흰색으로 나타나고 나머지 영역은 거의 검은색으로 나타남 → 계수값에 로그함수 적용




(a) 일반적 정규화 (b) 로그함수 후 정규화

```python
def calc_spectrum(complex):
    if complex.ndim==2:
        dst = abs(complex)                    # sqrt(re^2 + im^2) 계산해줌
    else:
        dst = cv2.magnitude(complex[:, :, 0], complex[:, :, 1]) # OpenCV의 경우
    dst = cv2.log(dst + 1)
    cv2.normalize(dst, dst, 0, 255, cv2.NORM_MINMAX)
    return cv2.convertScaleAbs(dst)
```

복소수 객체 행렬 사용 - 2차원 행렬

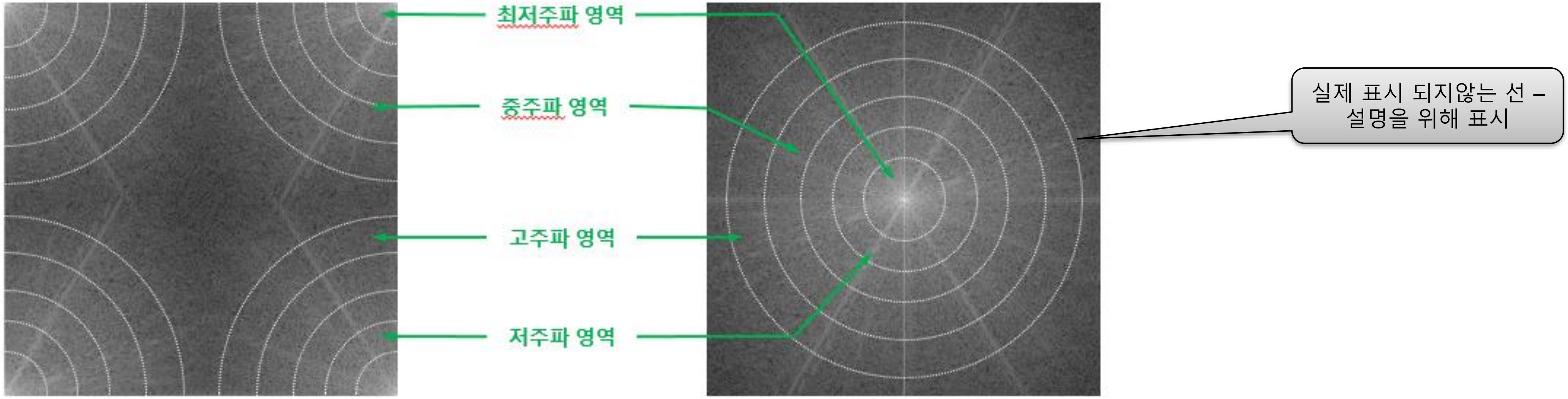
복소수를 2채널로 구성
- 3차원 행렬

log 함수는 0에서
무한대이기 때문에 1을 더함

윈도우 표시 위해 uchar 형변
환

## ❖ 셔플링(shuffling)/시프트(shift)

- 저주파 영역이 모서리 부분에 위치, 고주파 부분이 중심부에 위치
- 사각형의 각 모서리를 중심으로 원형의 밴드를 형성하여 주파수 영역 분포
  - 해당 주파수 영역 처리시 불편함 → 모양 변경 필요




DFT 수행 후 스펙트럼 영상

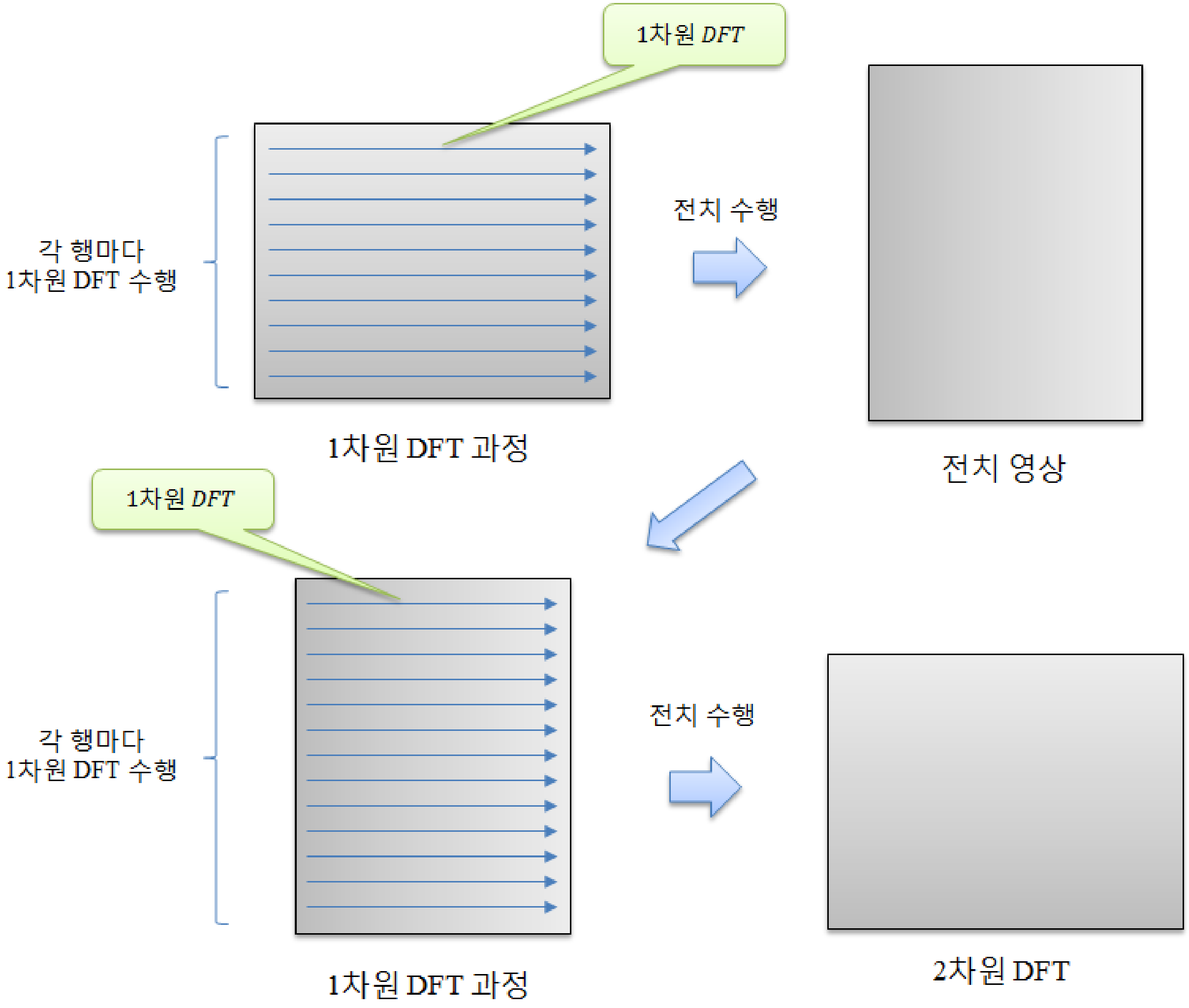
시프트 수행 후

## ❖ 스펙트럼의 셔플링은 원점 대칭인 주기함수이기 때문에 가능함

```
def fftshift(img):
    dst = np.zeros(img.shape, img.dtype)
    h, w = dst.shape[:2]
    cy, cx = h // 2, w // 2                              # 몫연산자
    dst[h-cy:, w-cx:] = np.copy(img[0:cy, 0:cx ])        # 1사분면-> 3사분면
    dst[0:cy, 0:cx ] = np.copy(img[h-cy:, w-cx:])        # 3사분면-> 1사분면
    dst[0:cy, w-cx:] = np.copy(img[h-cy:, 0:cx ])        # 2사분면-> 4사분면
    dst[h-cy:, 0:cx ] = np.copy(img[0:cy, w-cx:])        # 4사분면-> 2사분면
    return dst
```

# 2-d DFT 방법

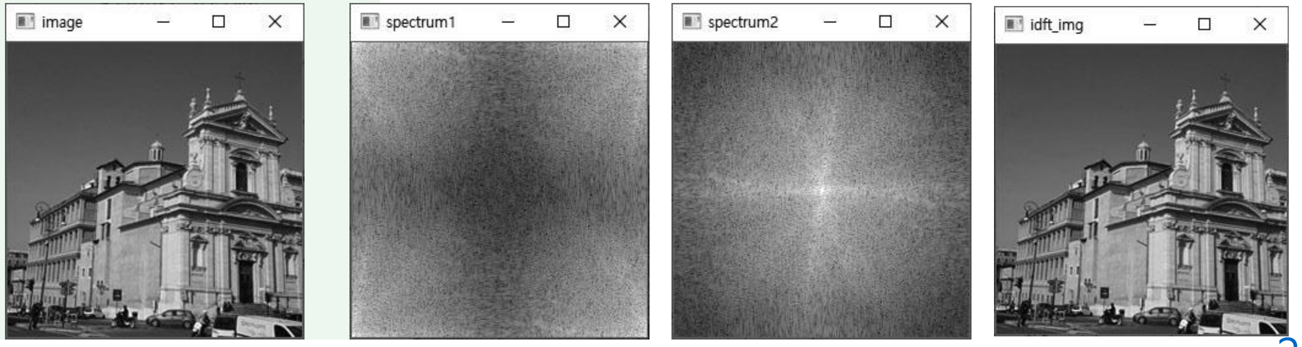
## ❖ 1차원 푸리에 변환 → 전치 → 1차원 푸리에 변환 → 전치



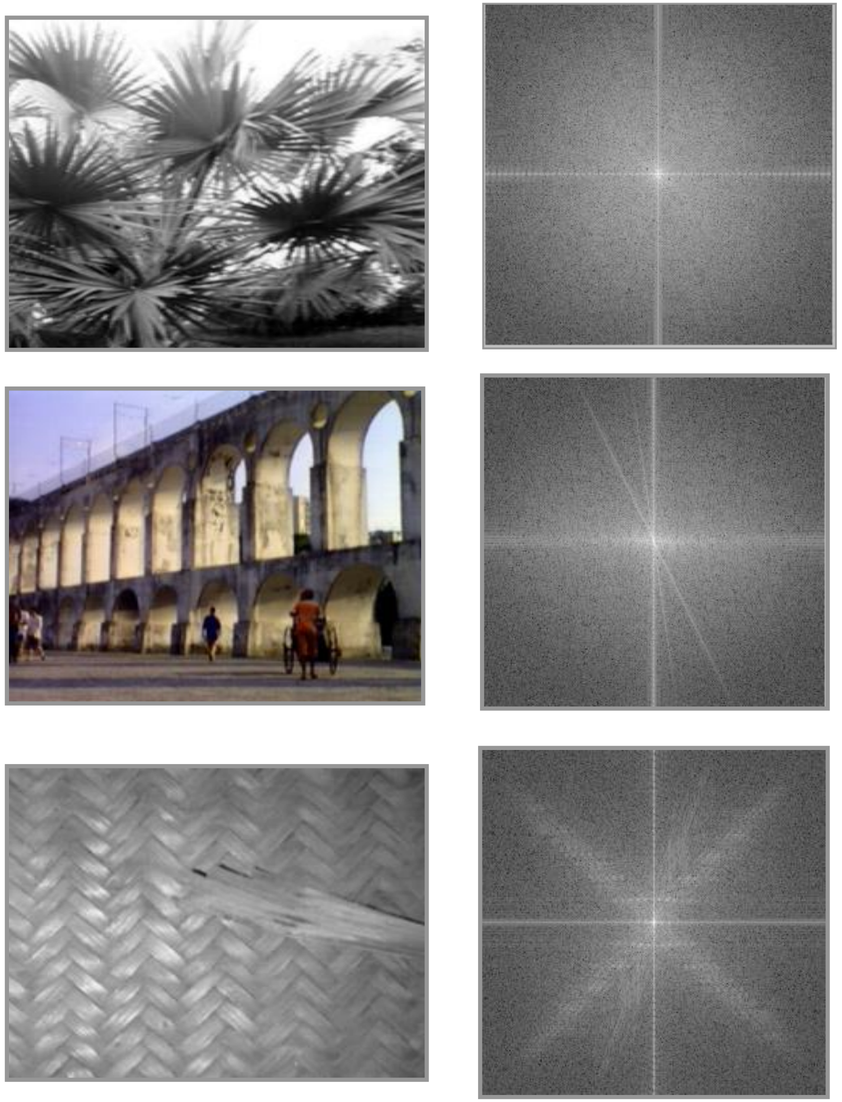
## 예제 9.2.3 2차원 이산 푸리에 변환 – 04.2d_dft.py

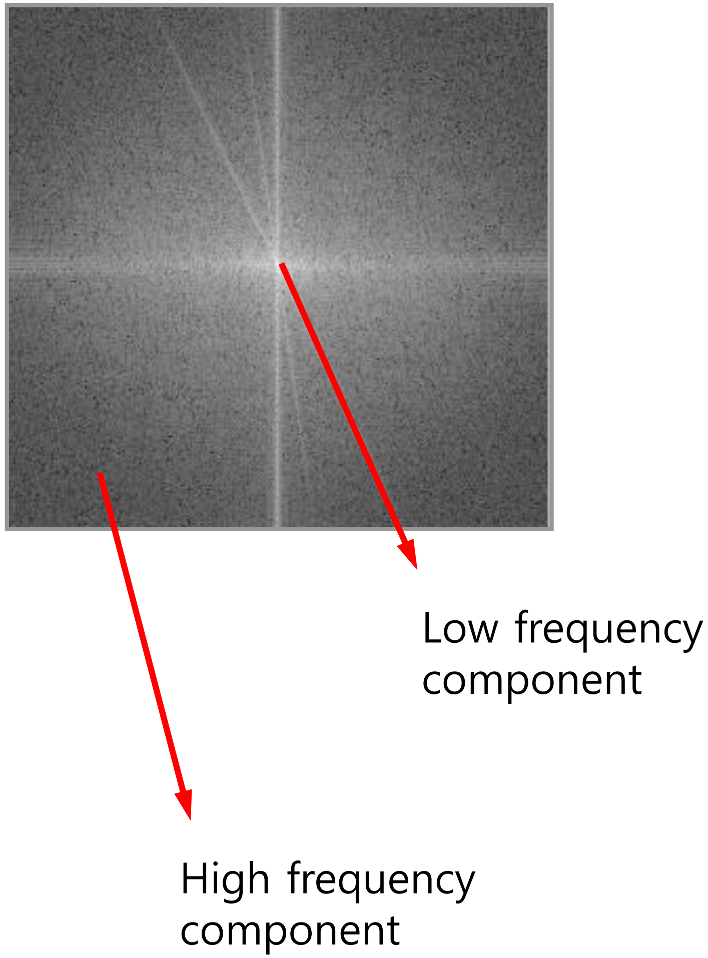
```python
import numpy as np, cv2
from Common.dft2d import dft, idft, calc_spectrum, fftshift

def dft2(image):
    tmp = [dft(row) for row in image]
    dst = [dft(row) for row in np.transpose(tmp)]
    return np.transpose(dst)                              # 전치 환

def idft2(image):
    tmp = [idft(row) for row in image]
    dst = [idft(row) for row in np.transpose(tmp)]
    return np.transpose(dst)                              # 전치 환원 후 반환

def ck_time(mode=0):                                      # 수행시간 체크 함수
    global stime                                          # 함수 내부에서 값 유지위해
    if (mode ==0 ):
        stime = time.perf_counter()
    elif (mode==1):
        etime = time.perf_counter()
        print("수행시간 = %.5f sec" % (etime - stime))

image = cv2.imread("images/dft_240.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상파일 읽기 에러")

ck_time(0)                                                # 시작 시간 체크
dft = dft2(image)                                         # 2차원 DFT 수행
spectrum1 = calc_spectrum(dft)                            # 주파수 스펙트럼 영상
spectrum2 = fftshift(spectrum1)                           # np.fft.fftshift() 사용 가능
idft = idft2(dft).real                                    # 2차원 IDFT 수행
ck_time(1)                                                # 종료 시간 체크

cv2.imshow("image", image)
cv2.imshow("spectrum1", spectrum1)
cv2.imshow("spectrum2", spectrum2)
cv2.imshow("idft_img", cv2.convertScaleAbs(idft))
```

```
Run: 04.2d_dft
C:\Python\python.exe D:/source/chap09/04.2d_dft.py
수행시간 = 157.06352 sec
```



## ❖ (예)





3

# 고속 푸리에 변환

# 고속 푸리에 변환의 필요성

## ❖ 이산 푸리에 변환

- 원본 신호의 한 원소에 곱해지는 기저 함수의 원소들은 원소 길이만큼 반복적으로 곱해짐
  - 신호가 커질수록 계산 속도는 기하급수적으로 증가

## ❖ 고속 푸리에 변환(Fast Fourier Transform)

- FFT는 DFT를 $\mathbf{O(n^2)}$보다 빠른 시간내에 구하는 방법
- 쿨리-튜키 알고리즘(Cooley-Tukey algorithm)을 주로 사용
  - 분할정복 알고리즘
    - 입력 벡터를 둘로 나누고 각각의 벡터에 대해 DFT를 계산한 후, $O(n)$ 시간에 합치는  알고리즘 : 알고리즘 복잡도 : $O(N \cdot \log N)$

# 원리

## ❖ 다항식은 계수들의 벡터로 표현 가능

- $A(x) = a_0x^0 + a_1x^1 + \cdots + a_{n-1}x^{n-1}$
- $w_{n,k} = e^{\frac{2k\pi}{n}i}, \qquad 0 \le k < n - 1$

- $w_n^k = (w_n^1)^k$
- $w_n^0 = 1$
- $w_n^{n/2} = e^{\pi i} = -1$, (Euler's identity)
- $w_n^{k+n} = w_n^k$
- $w_n^{k+n/2} = -w_n^k$

## ❖ DFT/IDFT

- $DFT(A) = DFT([a_0, a_1, \ldots a_{n-1}]) = [A(w_n^0), A(w_n^1), \ldots, A(w_n^{n-1})]$
- $IDFT([A(w_n^0), A(w_n^1), \ldots, A(w_n^{n-1})]) = [a_0, a_1, \ldots a_{n-1}]$

## ❖ $DFT_n$의 정의

$$DFT_n(A)_j = \sum_{i=0}^{n-1} a_i w^{ij}$$

$$\begin{bmatrix} DTF_4(A)_0 \\ DTF_4(A)_1 \\ DTF_4(A)_2 \\ DTF_4(A)_3 \end{bmatrix} = \begin{bmatrix} w^0 & w^0 & w^0 & w^0 \\ w^0 & w^1 & w^2 & w^3 \\ w^0 & w^2 & w^4 & w^6 \\ w^0 & w^3 & w^6 & w^9 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

❖ 짝수와 홀수의 순서를 바꾸어 다시 정리하면,

- $w_n^k = (w_n^1)^k$
- $w_n^0 = 1$
- $w_n^{n/2} = e^{\pi i} = -1$, (Euler's identity)
- $w_n^{k+n} = w_n^k$
- $w_n^{k+n/2} = -w_n^k$

$$\begin{bmatrix} DTF_4(A)_0 \\ DTF_4(A)_1 \\ DTF_4(A)_2 \\ DTF_4(A)_3 \end{bmatrix} = \begin{bmatrix} w^0 & \boxed{w^0} & \boxed{w^0} & w^0 \\ w^0 & w^1 & w^2 & w^3 \\ w^0 & w^2 & w^4 & w^6 \\ w^0 & w^3 & w^6 & w^9 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$= \begin{bmatrix} w^0 & w^0 & w^0 & w^0 \\ w^0 & w^2 & w^1 & w^3 \\ w^0 & w^4 & w^2 & w^6 \\ w^0 & w^6 & w^3 & w^9 \end{bmatrix} \begin{bmatrix} a_0 \\ a_2 \\ a_1 \\ a_3 \end{bmatrix} = \begin{bmatrix} w^0 & w^0 & w^0 w_0 & w^0 w_0 \\ w^0 & w^2 & w^1 w_0 & w^1 w_2 \\ w^0 & w^0 & w^2 w_0 & w^2 w_0 \\ w^0 & w^2 & w^3 w_0 & w^3 w_2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_2 \\ a_1 \\ a_3 \end{bmatrix}$$

❖ $DFT_2$를 빼서 정리

$$DFT_2 \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{bmatrix} w^0 & w^0 \\ w^0 & w^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & w^0 & 0 \\ 0 & 1 & 0 & w^1 \\ 1 & 0 & w^2 & 0 \\ 0 & 1 & 0 & w^3 \end{bmatrix} \begin{bmatrix} DFT_2 \begin{pmatrix} a_0 \\ a_2 \end{pmatrix} \\ DFT_2 \begin{pmatrix} a_1 \\ a_3 \end{pmatrix} \end{bmatrix}$$

# FFT의 원리

❖ **푸리에 변환의 수식을 짝수부와 홀수부로 분리**

$$G(k) = F_{even}(k) + F_{odd}(k) \cdot e^{-j2\pi k\frac{1}{2L}}$$

$$G(k)=\sum_{n=0}^{L-1} g[2n]\cdot e^{-j2\pi k\frac{2n}{2L}} + \sum_{n=0}^{L-1} g[2n+1]\cdot e^{-j2\pi k\frac{2n+1}{2L}}$$

$$G(k)=\left\{\sum_{n=0}^{L-1} g[2n]\cdot e^{-j2\pi k\frac{n}{L}}\right\}+\left\{\sum_{n=0}^{L-1} g[2n+1]\cdot e^{-j2\pi k\frac{n}{L}}\right\}\cdot e^{-j2\pi k\frac{1}{2L}}$$

$$G_{even}(k)=\sum_{n=0}^{L-1} g[2n]\cdot e^{-j2\pi k\frac{n}{L}}$$

$$G_{odd}(k)=\sum_{n=0}^{L-1} g[2n+1]\cdot e^{-j2\pi k\frac{n}{L}}$$

## ❖ 공통 지수부분에서 한 주기(L)를 더한 수식

$$e^{-j2\pi(k+L)\frac{n}{L}} = e^{-j2\pi k\frac{n}{L}} \cdot e^{-j2\pi\frac{Ln}{L}} = e^{-j2\pi k\frac{n}{L}} \cdot e^{-j2\pi n} = e^{-j2\pi k\frac{n}{L}}$$

$$\begin{aligned} G(k+L) &= G_{even}(k+L) + G_{odd}(k+L) \cdot e^{-j2\pi(k+L)\frac{1}{2L}} \\ &= G_{even}(k) + G_{odd}(k) \cdot e^{-j2\pi(k+L)\frac{1}{2L}} \end{aligned}$$

$$e^{-j2\pi(k+L)\frac{1}{2L}} = e^{-j2\pi k\frac{1}{2L}} \cdot e^{-j2\pi\frac{L}{2L}} = e^{-j2\pi k\frac{1}{2L}} \cdot e^{-j\pi} = -e^{-j2\pi k\frac{1}{2L}}$$

## ❖ 한 쌍의 $G(k)$ 으로 $G(k+L)$의 값 계산 가능 → 속도 향상

$$G(k+L) = G_{even}(k) - G_{odd}(k) \cdot e^{-j2\pi k\frac{1}{2L}}$$

# ❖ 짝수원소 홀수 원소 분리 및 계산

$$G(k)=G_{even}(k)+G_{odd}(k)\cdot e^{-j2\pi k\frac{1}{2L}}$$

$g[0]$, $g[2]$, $g[4]$, $g[6]$ → 짝수 원소 DFT → $G_{even}[0]$, $G_{even}[1]$, $G_{even}[2]$, $G_{even}[3]$

$g[1]$, $g[3]$, $g[5]$, $g[7]$ → 홀수 원소 DFT → $G_{odd}[0]$, $G_{odd}[1]$, $G_{odd}[2]$, $G_{odd}[3]$

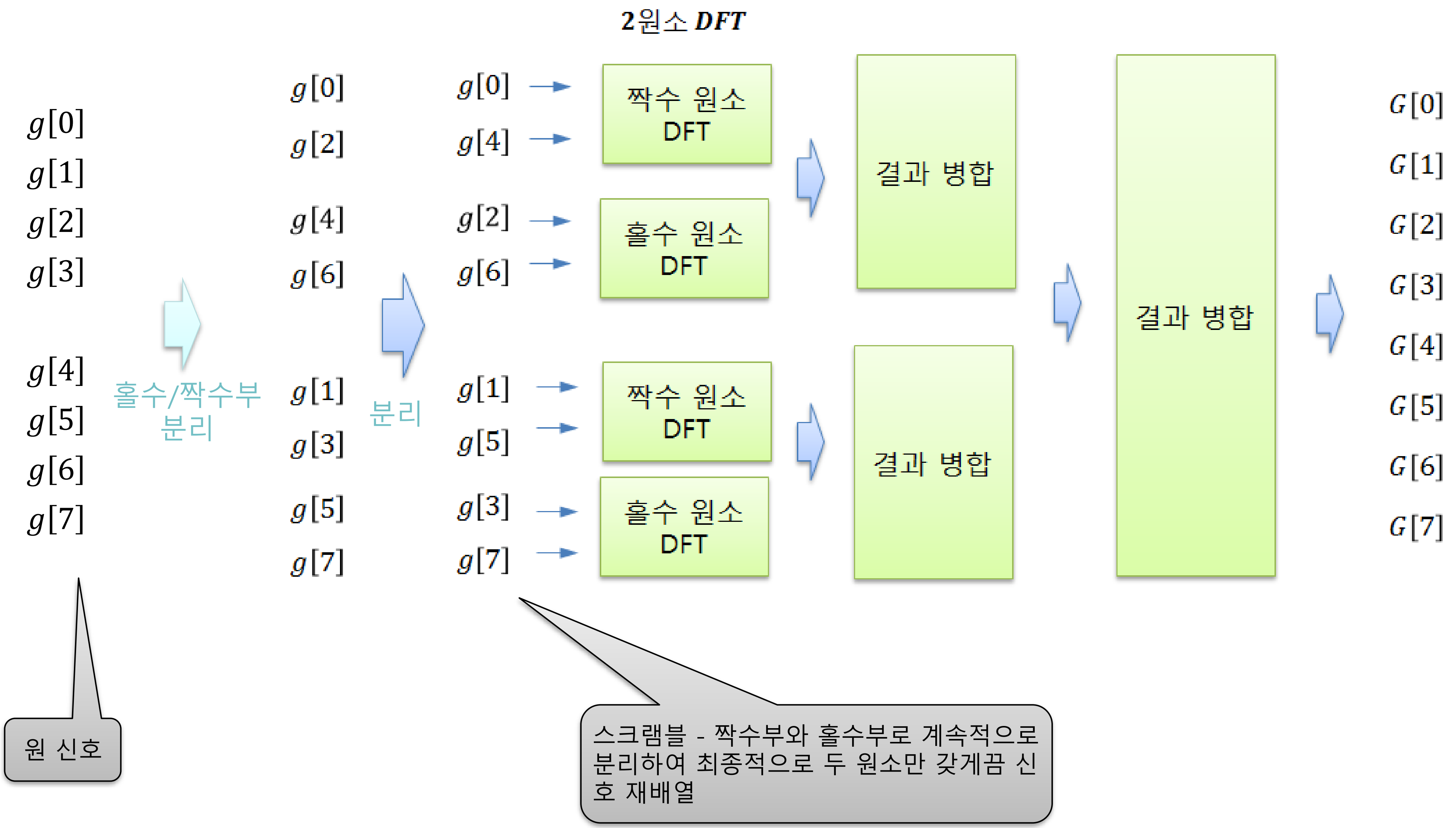
$$G[0] = G_{even}[0] + G_{odd}[0]\cdot e^{j2\pi\frac{0}{8}}$$
$$G[1] = G_{even}[1] + G_{odd}[1]\cdot e^{j2\pi\frac{1}{8}}$$
$$G[2] = G_{even}[2] + G_{odd}[2]\cdot e^{j2\pi\frac{2}{8}}$$
$$G[3] = G_{even}[3] + G_{odd}[3]\cdot e^{j2\pi\frac{3}{8}}$$

$$G[4] = G_{even}[0] - G_{odd}[0]\cdot e^{j2\pi\frac{4}{8}}$$
$$G[5] = G_{even}[1] - G_{odd}[1]\cdot e^{j2\pi\frac{5}{8}}$$
$$G[6] = G_{even}[2] - G_{odd}[2]\cdot e^{j2\pi\frac{6}{8}}$$
$$G[7] = G_{even}[3] - G_{odd}[3]\cdot e^{j2\pi\frac{7}{8}}$$

G[0]~G[3] 이용해서
G[4]~G[7] 계산 가능

$$G(k+L)=G_{even}(k)-G_{odd}(k)\cdot e^{-j2\pi k\frac{1}{2L}}$$

# ❖ 연속적 신호 분리 → 2원소로 분리

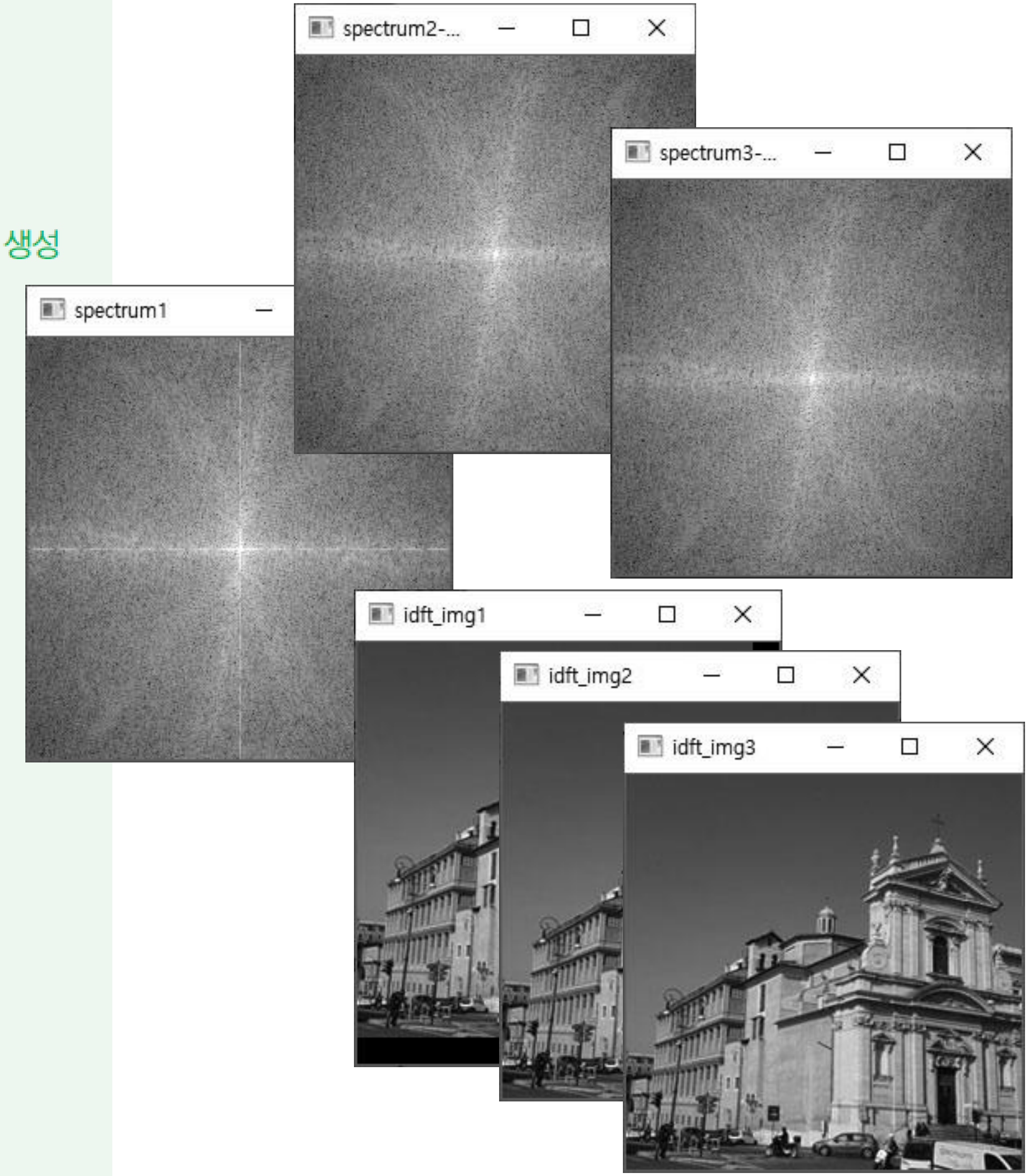


**예제 9.3.1** 고속 푸리에 변환 – 05.2d_fft.py

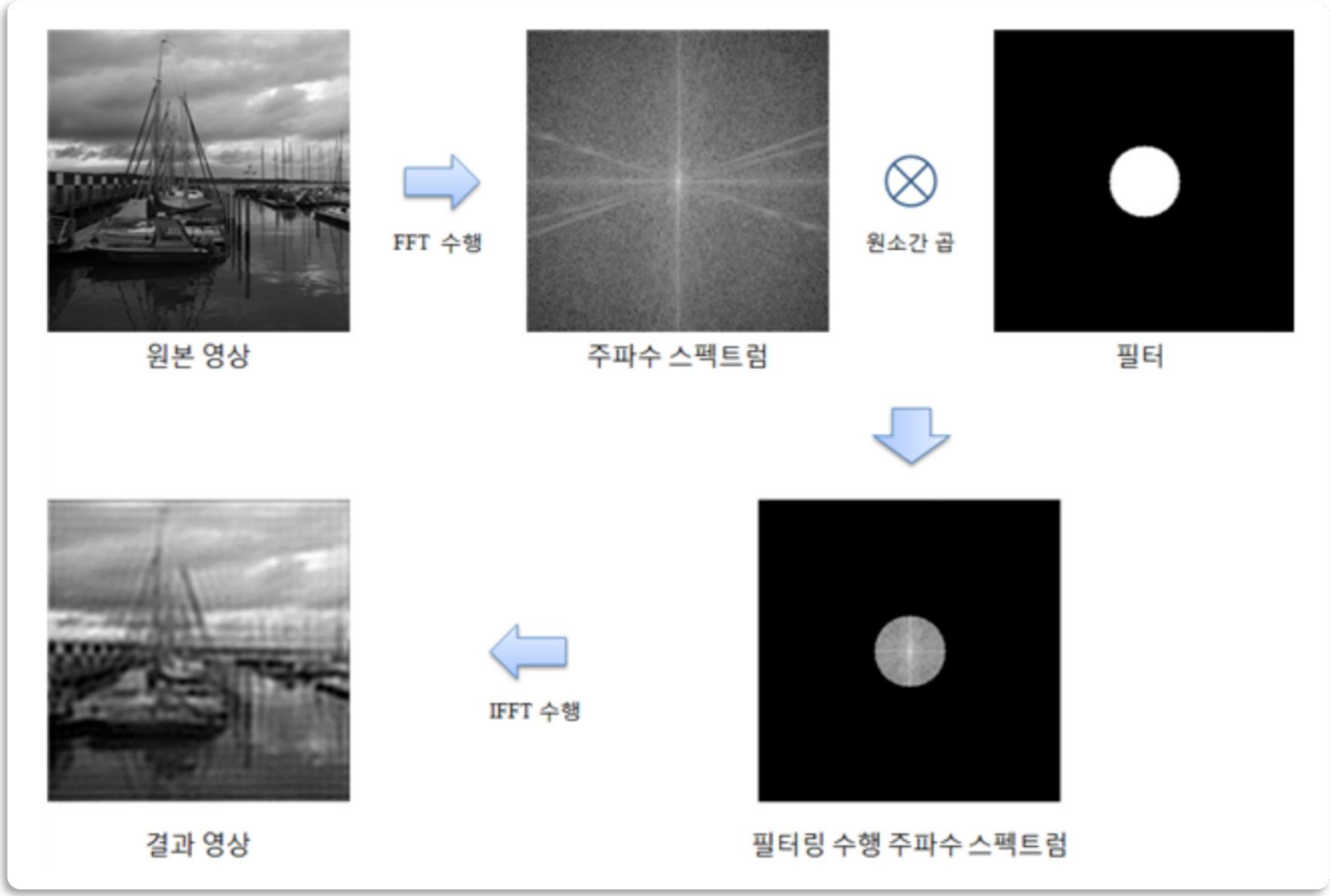
```
import numpy as np, cv2
from Common.dft2d import exp, calc_spectrum, fftshift          # dft 관련함수 임포트
from Common.fft2d import zeropadding                           # 영삽입 함수 임포트

def butterfly(pair, L, N, dir):                                # 버터플라이 수행 함수
    for k in range(L):
        Geven, Godd = pair[k], pair[k + L]
        pair[k]      = Geven + Godd * exp(dir * k / N)          # 짝수부
        pair[k + L]  = Geven - Godd * exp(dir * k / N)          # 홀수부

def parring(g, N, dir, start=0, stride=1):                     # 2원소까지 재귀 통한 분리&합성
    if N == 1: return [g[start]]
    L = N // 2
    sd = stride * 2
    part1 = parring(g, L, dir, start, sd)                      # 홀수 신호 재귀 분리
    part2 = parring(g, L, dir, start + stride, sd)             # 짝수 신호 재귀 분리
    pair = part1 + part2                                       # 재귀 결과 병합
    butterfly(pair, L, N, dir)                                 # 버터플라이 수행
    return pair
```

```python
def fft(g):                                              # 1차원 fft 수행
    return pairing(g, len(g), 1)

def ifft(g):                                             # 1차원 ifft 수행
    fft = pairing(g, len(g), -1)
    return [v / len(g) for v in fft]                     # 실수부만 반환

def fft2(image):
    pad_img = zeropadding(image)                         # 영삽입
    tmp = [fft(row) for row in pad_img]
    dst = [fft(row) for row in np.transpose(tmp)]        # 전치 후 1차원 푸리에 변환
    return np.transpose(dst)                             # 전치 환원 후 반환

def ifft2(image):
    tmp = [ifft(row) for row in image]
    dst = [ifft(row) for row in np.transpose(tmp)]       # 전치후 1차원 푸리에 변환
    return np.transpose(dst)

image = cv2.imread(“images/dft_240.jpg”, cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception(“영상파일 읽기 에러”)
```

```
dft1 = fft2(image)                                        # 저자 구현 fft 함수
dft2 = np.fft.fft2(image)                                 # numpy fft 함수
dft3 = cv2.dft(np.float32(image), flags=cv2.DFT_COMPLEX_OUTPUT)  # OpenCV fft 함수

spectrum1 = calc_spectrum(fftshift(dft1))          # 시프트후 주파수 스펙트럼 영상 생성
spectrum2 = calc_spectrum(fftshift(dft2))
spectrum3 = calc_spectrum(fftshift(dft3))

idft1 = fft2(dft1).real                            # 저자 구현 fft 역변환
idft2 = np.fft.ifft2(dft1).real                    # numpy fft 역변환
idft3 = cv2.idft(dft3, flags=cv2.DFT_SCALE)[:,:,0] # OpenCV fft 역변환

print("user 방법 변환 행렬 크기:", dft1.shape)       # 푸리에 변환 결과 행렬 형태
print("np.fft 방법 변환 행렬 크기:", dft2.shape)
print("cv2.dft 방법 변환 행렬 크기:", dft3.shape)

cv2.imshow("spectrum1", spectrum1)                 # 주파수 스펙트럼 영상
cv2.imshow("spectrum2-np.fft", spectrum2)
cv2.imshow("spectrum3-OpenCV", spectrum3)
cv2.imshow("idft_img1", cv2.convertScaleAbs(idft1))  # 역변환 후 환원 영상
cv2.imshow("idft_img2", cv2.convertScaleAbs(idft2))
cv2.imshow("idft_img3", cv2.convertScaleAbs(idft3))
cv2.waitKey(0)
```



# 주파수 영역 필터링

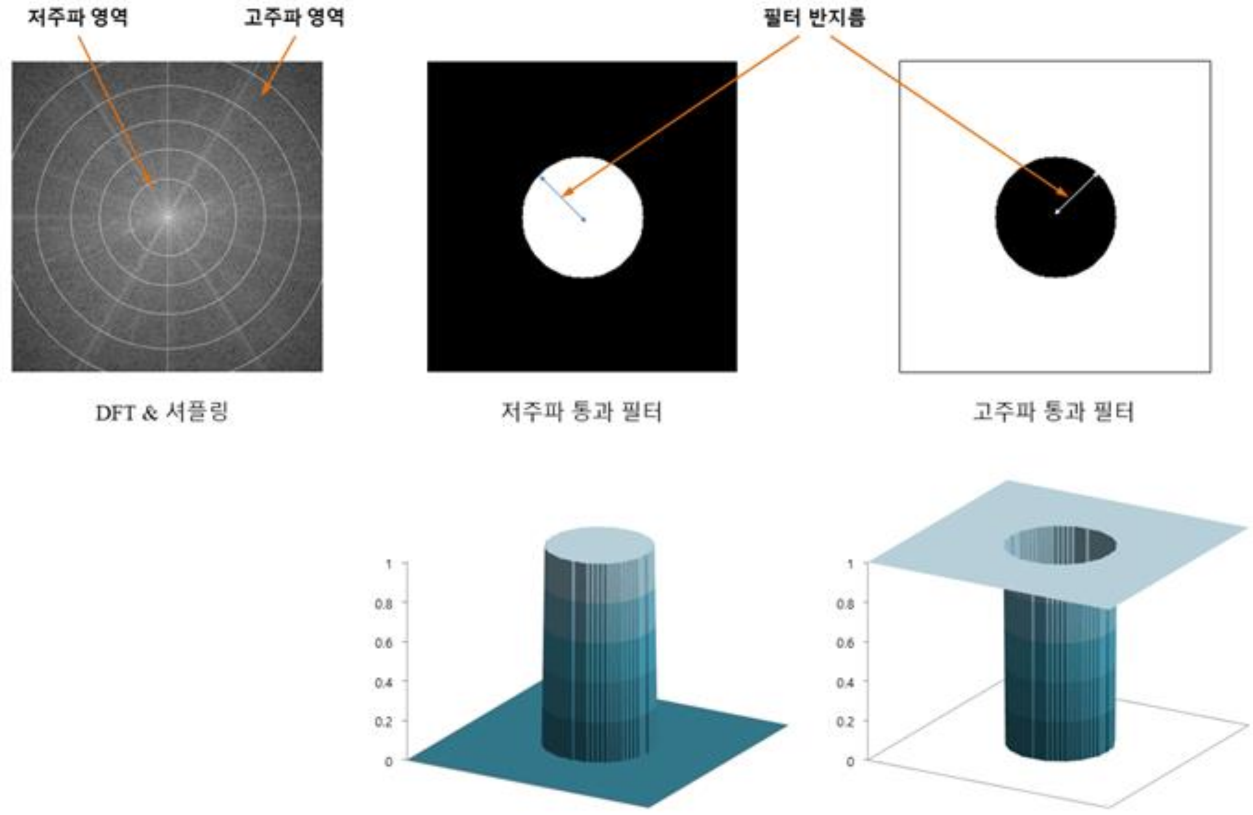


## ❖ 저주파 통과 필터링

- DFT 변환 영역에서 저주파 영역의 계수들은 통과, 고주파 영역의 계수 차단

## ❖ 고주파 통과 필터링

- 고주파 영역의 계수들을 통과시키고 저주파 영역의 계수들 차단



# Butterworth & Gaussian filter

## ❖ 대역 통과 필터

- 특정한 대역에서 급격하게 값을 제거하기 때문에 결과 영상의 화질 저하
- 객체의 경계부분 주위로 잔물결 같은 무늬(ringing pattern) 나타남

## ❖ 해결방법

- 필터 원소값을 차단 주파수에서 급격하게 0으로하지 않고 완만한 경사 이루도록 구성
- 버터워즈 필터(Butterworth filter)나 가우시안 필터(Gaussian filter)

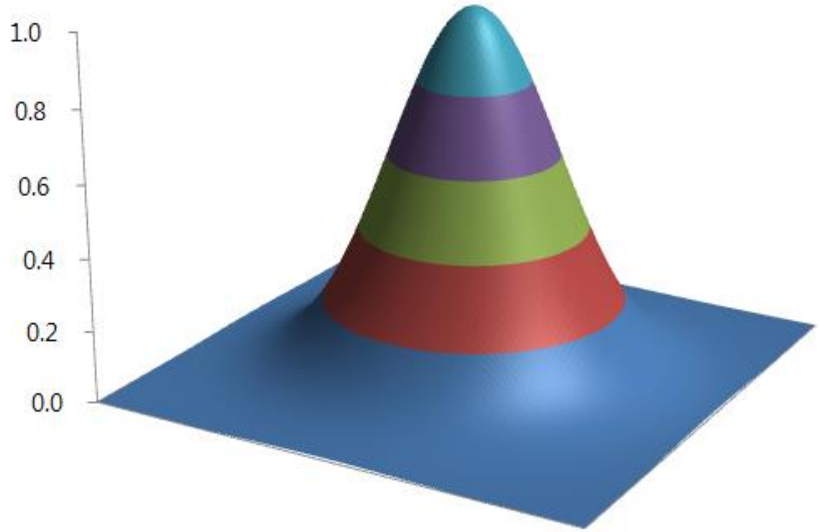
## ❖ 가우시안 필터

- 필터 원소의 구성을 가우시안 함수의 수식 분포를 갖게 함으로써 차단 주파수 부분을 점진적으로 구성한 것

$$f(x, y) = exp\left(-\frac{dx^2 + dy^2}{2R^2}\right),$$

$$dx = x - center.x$$
$$dy = y - center.y$$
$R$ : 주파수 차단 반지름

필터 계수를 밝기로 표현




필터 계수를 3차원 표현

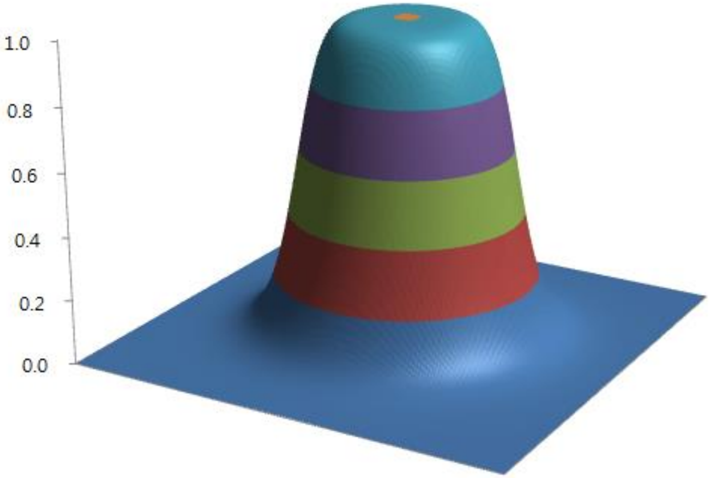
## ❖ 버터워즈 필터

- 차단 주파수 반지름 위치(*R*)와 지수의 승수인 *n* 값에 따라서 차단 필터의 반지름과 포물선의 곡률 결정

$$f(x, y) = -\frac{1}{1+\left(\frac{\sqrt{dx^2+dy^2}}{R}\right)^{2n}},$$

$$dx = x - center.x$$
$$dy = y - center.y$$
$R$ : 주파수 차단 반지름

필터 계수를 밝기로 표현




필터 계수를 3차원 표현

# 4

# 이산 코사인 변환

# 이산 코사인 변환

## ❖ 이산 코사인 변환(DCT: Discrete Cosine Transform)

- 1974년 미국의 텍사스 대학(University of Texas)에서 라오 교수 팀이 발표한 직교변환에 관한 논문
  - 영상 신호의 에너지 집중 특성이 뛰어나서 **영상 압축에 효과적**인 주파수 변환 방법을 찾는 것이 목표였음
- 이산 푸리에 변환(DFT)에서 실수부만 취하고, 허수부분 제외

$$F(k)=C(k)\cdot\sum_{n=0}^{N-1}g[n]\cdot\cos\left(\frac{(2n+1)k\pi}{2N}\right)$$

$$g[n]=\sum_{k=0}^{N-1}C(k)\cdot F(k)\cdot\cos\left(\frac{(2n+1)k\pi}{2N}\right)$$

$$\text{단}, k=0,\cdots,N-1,\quad C(k)=\begin{cases}\sqrt{\frac{1}{N}}, & \text{if } k=0\\ \sqrt{\frac{2}{N}}, & \text{if } k\neq 0\end{cases}$$

## ❖ 2차원 DCT

$$F(k,l)=C(k)\cdot C(l)\cdot \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} g[n,m]\cdot \cos\left(\frac{(2n+1)k\pi}{2N}\right)\cdot \cos\left(\frac{(2m+1)l\pi}{2M}\right)$$

$$g[n,m]=\sum_{k=0}^{N-1}\sum_{l=0}^{M-1} C(k)\cdot C(l)\cdot F(k,l)\cdot \cos\left(\frac{(2n+1)k\pi}{2N}\right)\cdot \cos\left(\frac{(2n+1)l\pi}{2M}\right)$$

$$\text{단, } k=0,\cdots,N-1, \quad C(k)=\begin{cases}\sqrt{\frac{1}{N}}, & \text{if } k=0\\ \sqrt{\frac{2}{N}}, & \text{if } k\neq 0\end{cases}$$

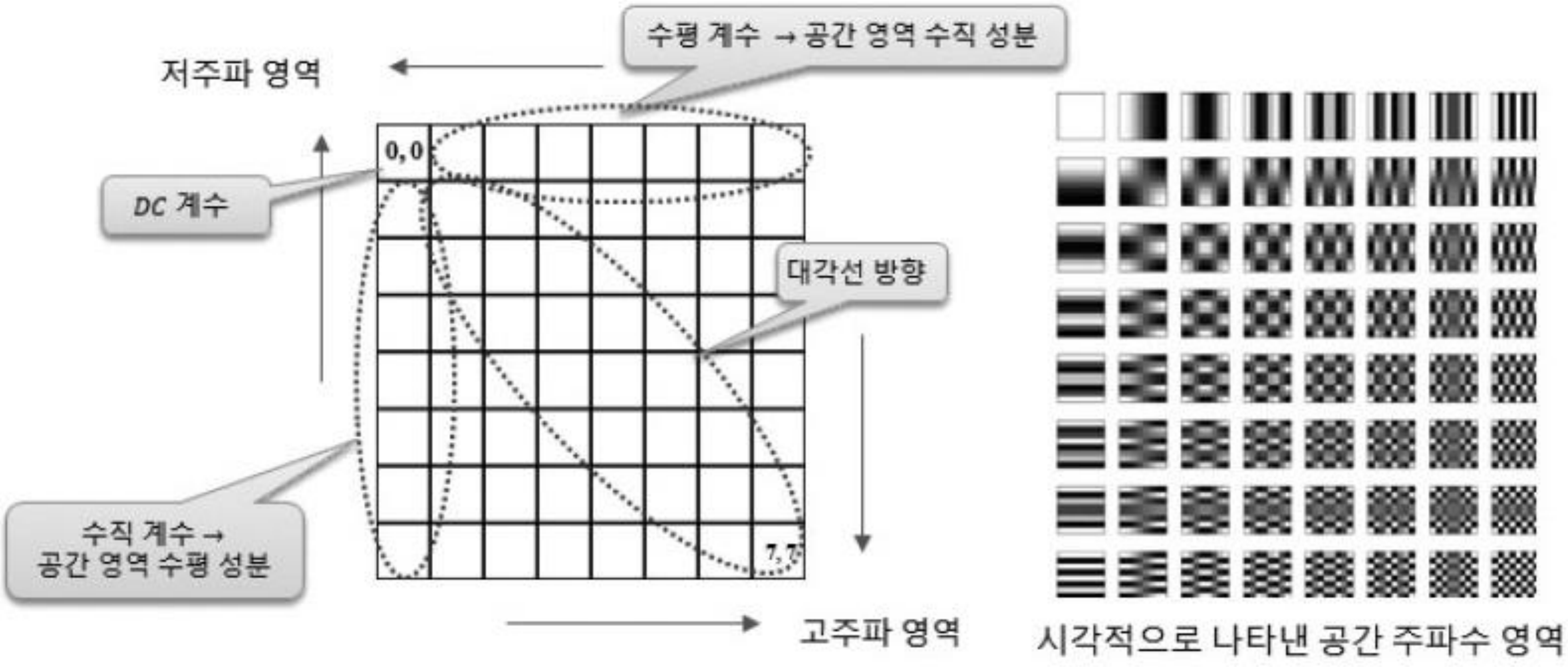
$$l=0,\cdots,M-1, \quad C(l)=\begin{cases}\sqrt{\frac{1}{N}}, & \text{if } l=0\\ \sqrt{\frac{2}{N}}, & \text{if } l\neq 0\end{cases}$$

## ❖ 전체 영상을 한 번에 변환시키는 것이 아니라 영상을 작은 블록으로 나누어서 수행

- 블록의 크기를 키울수록 압축의 효율이 높아지지만, 변환의 구현이 어려워지고 속도
- 일반적으로 8×8 크기 사용

# ❖ DCT 계수의 주파수 특성

- 왼쪽 상단으로 갈수록 저주파 영역이며, 오른쪽 하단으로 갈수록 고주파영역




〈그림 9.5.1〉 Forward DCT 변환 계수의 주파수 성분