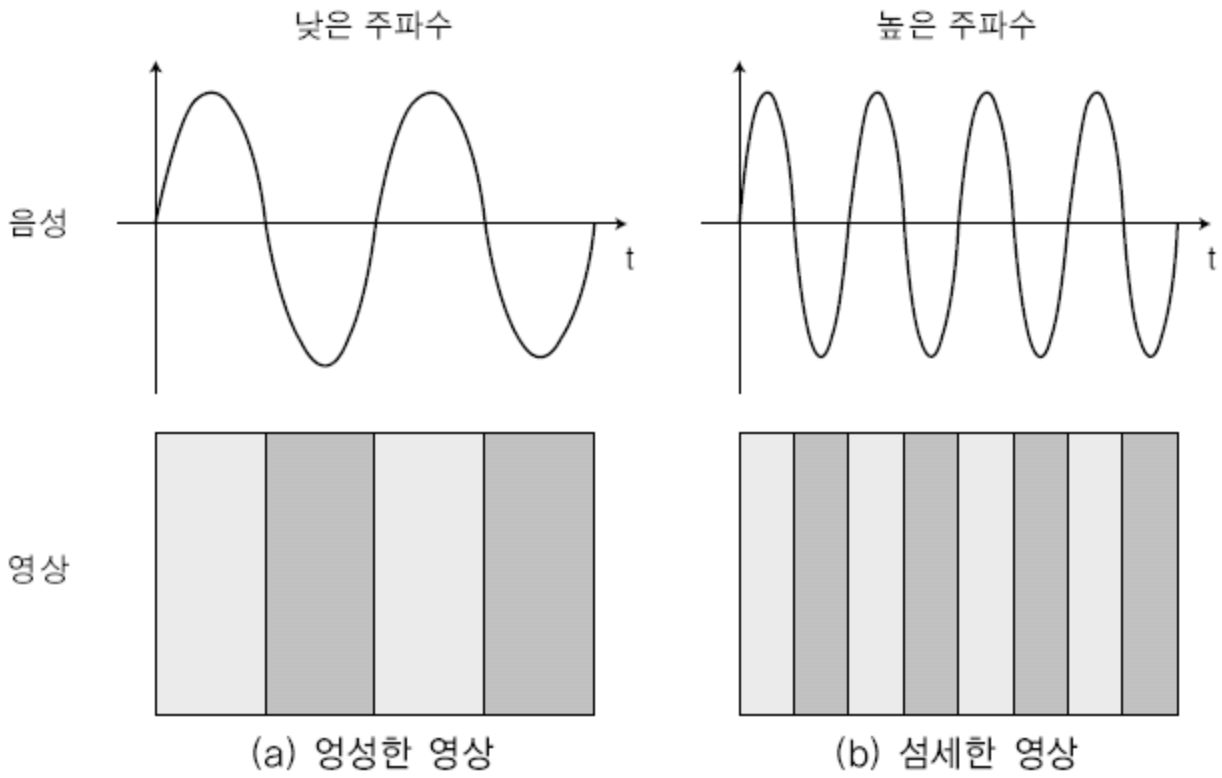
13장 영상 변환

- 영상 변환의 개요
- 주파수 변환
- 주파수 영역에서의 필터링
- 웨이브렛 변환

학습목표

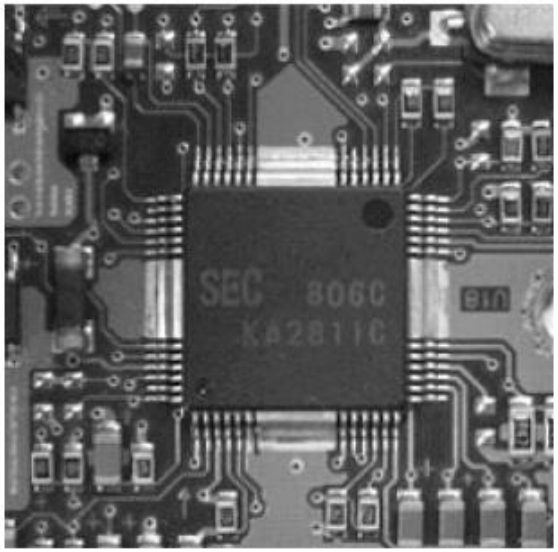
- ✓ 주파수의 개념을 이해한다.
- ✓ 푸리에 변환과 고속 푸리에 변환을 소개한다.
- ✓ 이산 코사인 변환의 특징을 이해한다.
- ✓ 주파수 영역에서 필터링의 특징을 공부한다.
- ✓ 웨이브렛 변환의 개념을 소개한다.

- 주파수
  - 영상에서 화소 밝기의 변화 정도를 나타내는 것은 화소 값의 변화율
  - 주파수는 밝기가 얼마나 빨리 변화하는가에 따라서 고주파와 저주파로 분류



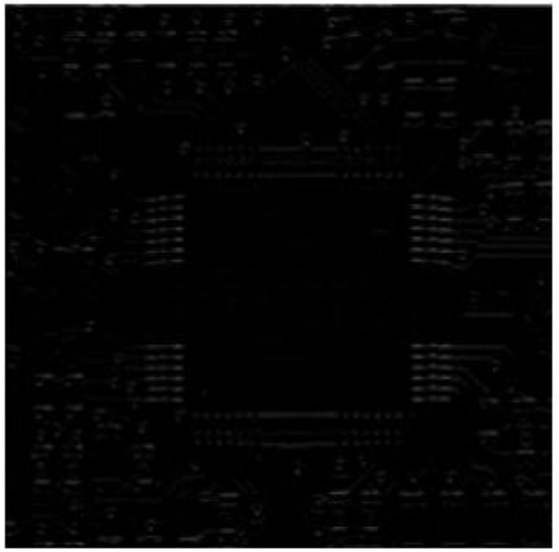
[그림 13-1] 저주파 영상과 고주파 영상의 개념

영상 변환의 개요(계속)



(a) 원본 영상

(b) 고역을 낮춤
(섬세한 부분이 없어져 둔해진다.)



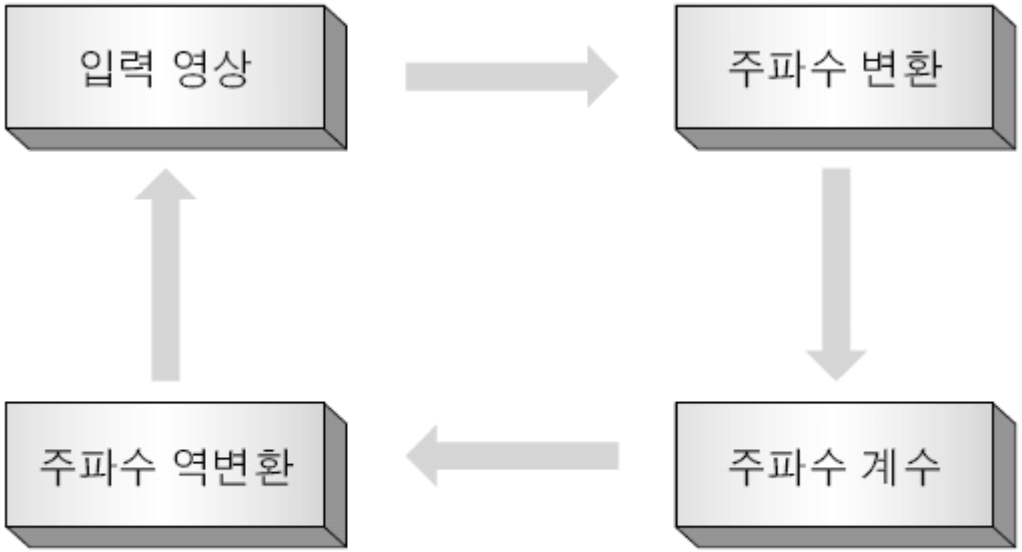
(c) 저역을 낮춤
(엉성한 부분이 없어져 에지가 강조된다.)

[그림 13-2] 영상의 주파수 영역에서 주파수 처리

- 영상을 공간 주파수 영역으로 변환하면 저주파와 고주파 성분으로 분리됨
  - 높은 주파수 성분을 낮추면 섬세한 부분이 사라지고, 부드럽고 엉성한 영상으로 변함.
  - 낮은 주파수 성분을 낮추면 엉성한 부분이 사라지면서 섬세한 부분에 해당하는 경계가 강조됨.

주파수 변환

- 공간 영역 형태의 영상을 주파수 영역 형태의 기본 주파수로 분리하는 것
- 정규적인 변환이 성립하려면 역변환도 성립되어야 함
  - 주파수 변환에는 주파수 형태의 영상을 공간 형식으로 변환하는 역주파수 변환이 반드시 있어야 함.

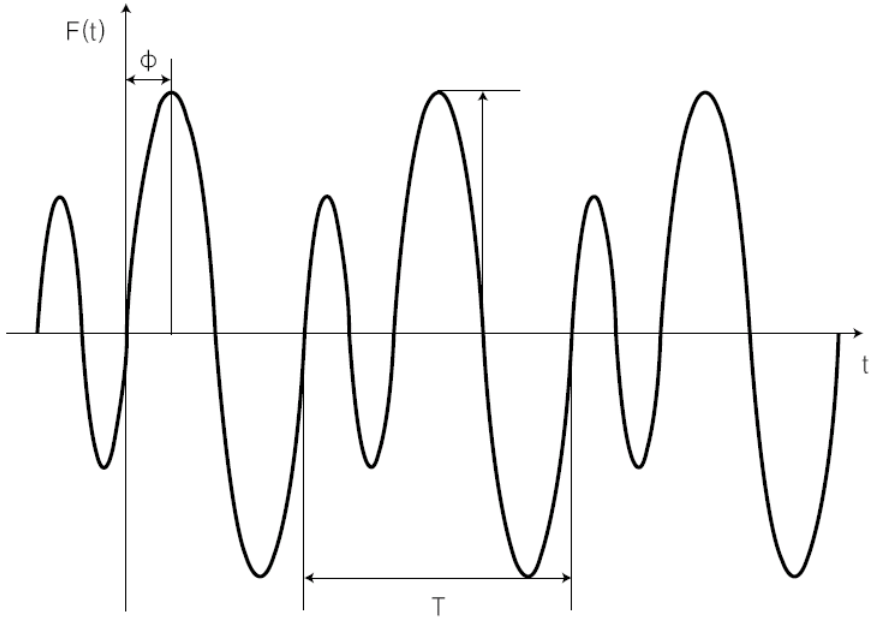


[그림 13-3] 주파수 변환의 개념

푸리에 변환(Fourier Transform)

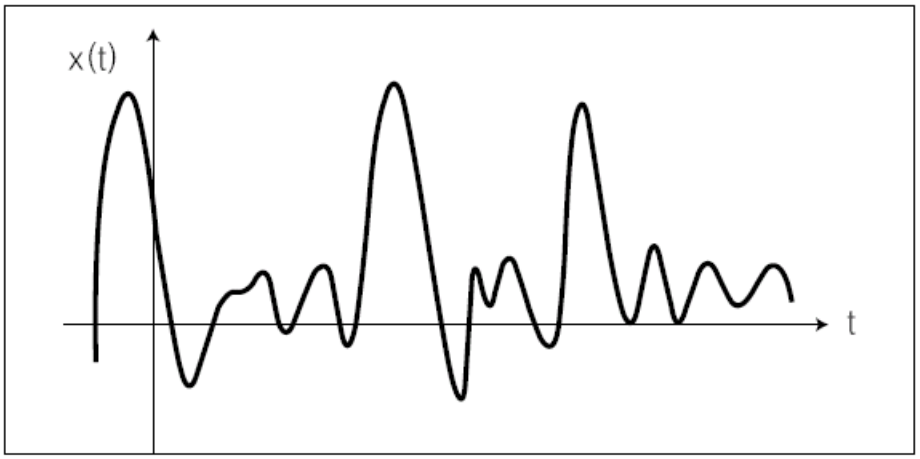
- 푸리에 변환(Fourier Transform)
  - 주파수 영역으로 변환하는 가장 일반적인 방법
  - 주기성이 있는 신호는 연속된 정현파의 조합으로 표현

- 신호를 구성하는 세 가지 요소
  - 주기 T: 반복되는 시간
  - 진폭 A: 파형의 크기. 0에서 양의 최대 높이까지의 거리
  - 위상 Φ : 파형의 시작이 얼만큼 지연되고 선행되었는지를 나타내는 시간 차이



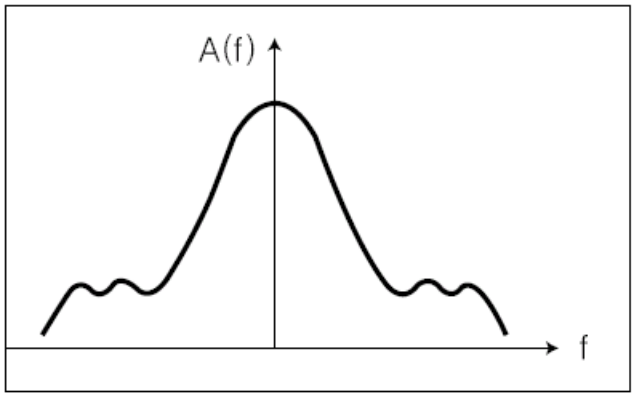
[그림 13-4] 신호의 구성요소

푸리에 변환(변환)

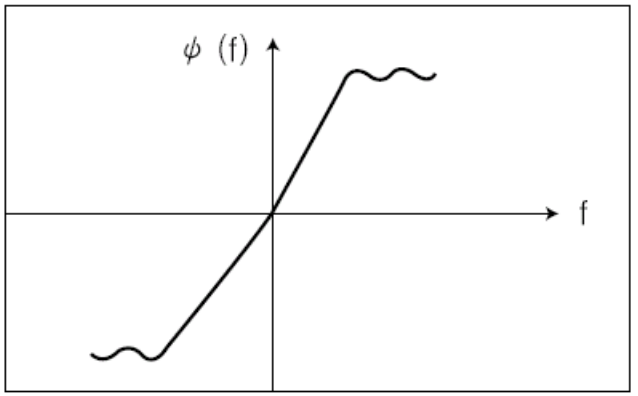


(a) 시간 영역에서의 신호 파형

푸리에 변환



(b) 주파수 영역에서의 신호 크기



(c) 주파수 영역에서의 신호 위상

[그림 13-6] 푸리에 변환을 이용한 주파수 영역으로의 변환

푸리에 변환(계속)

- 연속 푸리에 변환
  - 연속적인 시간 영역의 신호를 주파수 영역으로 변환하는 것.

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

  - 2차원 연속 푸리에 변환 공식

$$\Im^{-1}\{G(f)\} = g(t) = \int_{-\infty}^{\infty} G(f)e^{j2\pi ft}df$$

$$\Im\{g(x,y)\} = G(f_x, f_y) = \iint_{-\infty} g(x,y)e^{-j2\pi(f_x x + f_y y)}dxdy$$

$$\Im^{-1}\{G(f_x, f_y)\} = g(x,y) = \iint_{-\infty} G(f_x, f_y)e^{j2\pi(f_x x + f_y y)}df_x df_y$$

푸리에 변환(계속)

- 이산 푸리에 변환
  - 디지털 영상은 아날로그 신호가 아니고 디지털 데이터이므로, 연속 푸리에 변환에 직접적으로 적용할 수 없음.
  - 이산 푸리에 변환(**Discrete Fourier Transformation**)은 디지털 신호를 주파수 영역으로 변환해 줌.
  - 이산 푸리에 변환 공식: 연속 푸리에 변환의 적분을 합(Sum)으로 변경

$$G(\hat{f}) = \frac{1}{N}\sum_{n=0}^{N-1} g[n] e^{-j2\pi \hat{f}\frac{n}{N}}$$

$$g[n] = \sum_{\hat{f}=0}^{N-1} G(\hat{f}) e^{j2\pi \hat{f}\frac{n}{N}}$$

$$G(\hat{f}_n, \hat{f}_m) = \frac{1}{MN}\sum_{n=0}^{N-1}\sum_{m=0}^{M-1} g[n,m] e^{-j2\pi(\hat{f}_n n/N + \hat{f}_m m/M)}$$

$$g[n,m] = \frac{1}{MN}\sum_{\hat{f}_n=0}^{M-1}\sum_{\hat{f}_m=0}^{N-1} G(\hat{f}_n, \hat{f}_m) e^{j2\pi(\hat{f}_n n/N + \hat{f}_m m/M)}$$

고속 푸리에 변환(Fast Fourier Transformation: FFT)

- 이산 푸리에 변환은 복잡하고 연산량이 많아 하드웨어를 구현할 때 처리 속도가 늦어진다는 단점이 있음
- 고속의 푸리에 변환(**Fast Fourier Transformation: FFT**)은 이산 푸리에 변환 공식에서 반복 계산을 제거하면 변환을 빠르게 수행할 수 있음.
- **1차원 DFT**를 두 번 수행하는 **DFT**의 분리성을 이용하여 **2차원 DFT**를 수행

$$G(\hat{f}_n, \hat{f}_m) = \frac{1}{MN}\sum_{n=0}^{N-1}\left(e^{-j2\pi \hat{f}_m m/N} \left\{ \sum_{m=0}^{M-1} g[n,m] e^{-j2\pi \hat{f}_n n/M} \right\} \right)$$

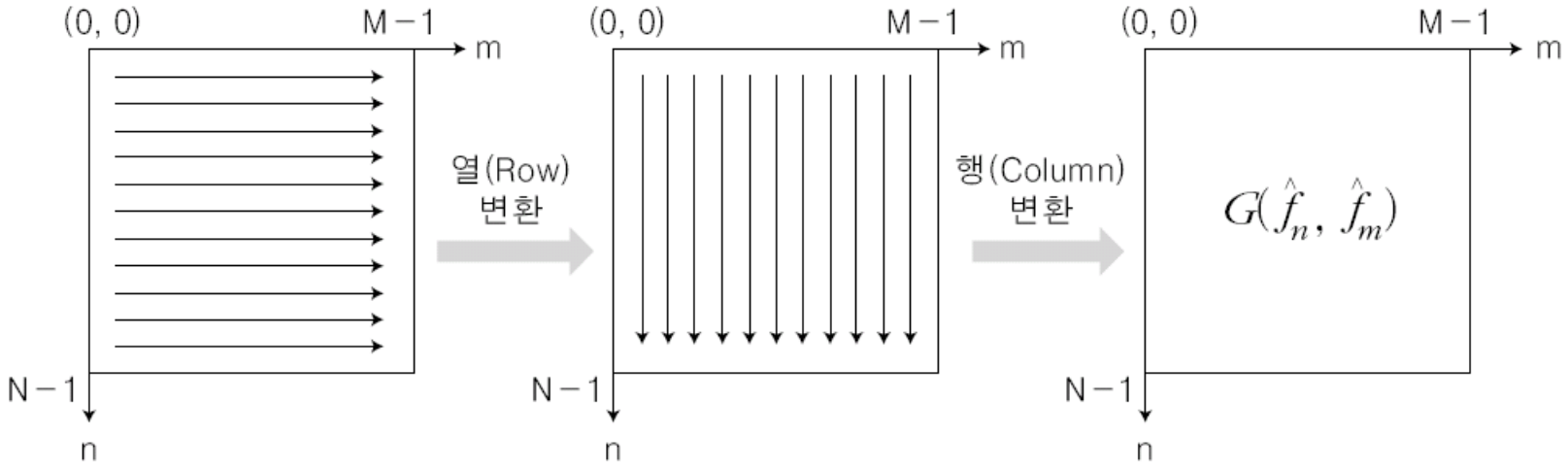
- 분리성을 이용하면 1차원 푸리에 변환이 두 단계 적용됨

$$G(n, \hat{f}_m) = \frac{1}{M}\sum_{m=0}^{M-1} g[n,m] e^{-j2\pi \hat{f}_n n/M}$$

$$G(\hat{f}_n, \hat{f}_m) = \frac{1}{N}\sum_{n=0}^{N-1} G(n, \hat{f}_m) e^{-j2\pi \hat{f}_m m/N}$$

고속 푸리에 변환(계속)

- 분리성 때문에 1차원 DFT는 효과적으로 1차원 FFT를 적용할 수 있어 계산량을 크게 줄일 수 있고,
- 하드웨어 구현이 용이해짐. 결국, 2차원 FFT가 수행되는 것



[그림 13-7] 2차원 FFT의 수행

순방향 고속 푸리에 변환

- **FFT**를 영상에 적용하려면 필수적으로 영상의 크기도 **2**의 지수 승이어야 함**(**예를 들어**, N=2^j, M=2^k).**
- 영상의 크기가 **2**의 지수 승이 아니라면 **0**의 값을 삽입하여 강제적으로 **2**의 지수 승을 만듦**. 1**차원 **FFT**는 두 단계로 구현됨**.**
  - 첫 번째 단계**:** 스크램블링
    - 재귀적인 **DFT** 계산 주기와 맞추려고 데이터를 적절히 재배치

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk},\ k=0,1\cdots,N-1$$

  - 두 번째 단계**:** 버터플라이 함수 적용
    - 데이터를 점**(Pointer)**의 집합으로 나눠 이웃한 점의 **DFT** 변환 수행

$$W_N^{nk}=e^{-j2\pi(nk/N)}$$

  - 결국**,** 다음과 같이 정리할 수 있음**.**

$$X(k)=G(k)+W_N^kH(k)$$

영상 스펙트럼

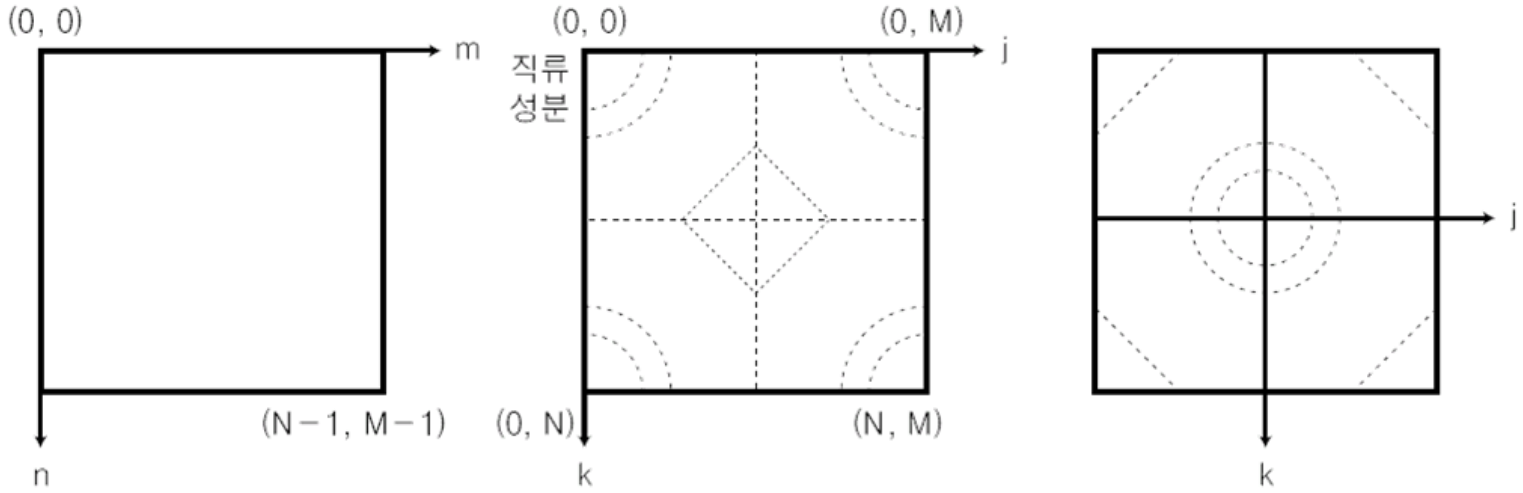
- 스펙트럼(Spectrum) 영상
  - 디지털 영상이 이산 푸리에 변환으로 주파수 영역 영상으로 변환되는 것
- 스펙트럼 영상을 나타낼 때 구한 주파수 데이터의 동적 범위가 너무 넓음
- 스펙트럼의 상용 대수식

$$D(u,v) = c\log[1+|G(\hat{f}_n, \hat{f}_m)|]$$

  - 1을 더하는 이유는 화소 값이 0일 때를 고려한 것임

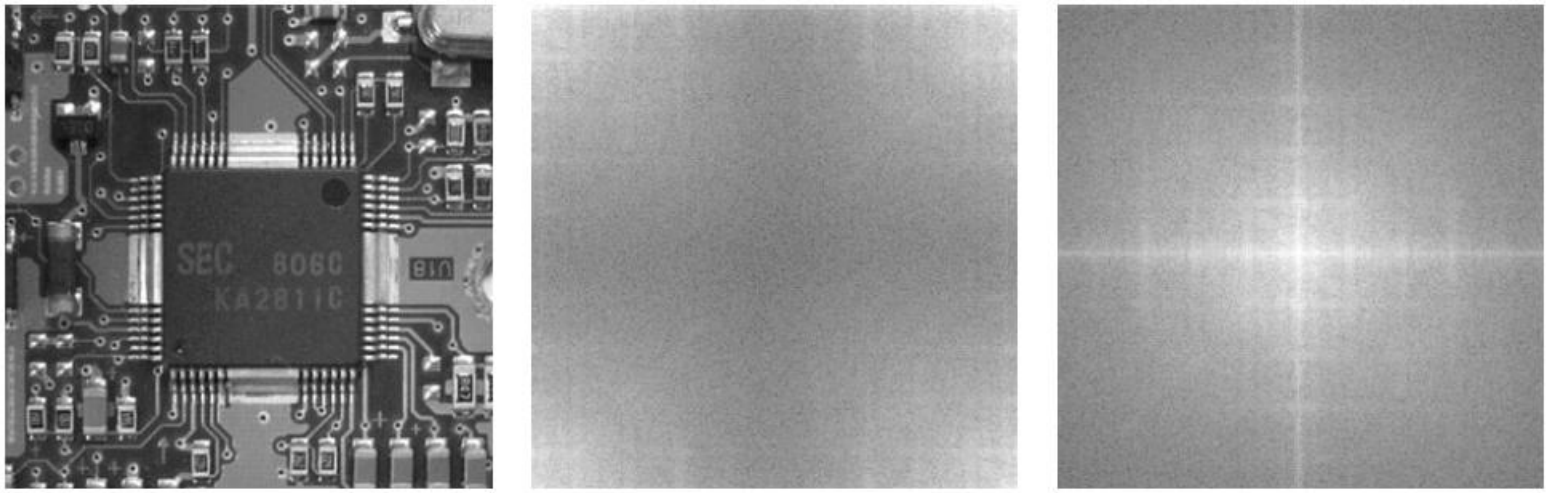
- 스펙트럼 영상에는 푸리에 변환의 결레 대칭성으로 대칭 성분이 발생하여 사각형의 네 구석에 직류 성분이 있게 됨.
- 샤플링(Shuffling)은 푸리에 변환의 주기성을 이용하여 주파수 4분 면을 영상의 가운데로 이동시키는 것으로, 좀더 쉽게 해석해 주는 역할 함.

영상 스펙트럼(계속)



(a) 입력 영상 (b) 푸리에 변환 (c) 셔플링

[그림 13-11] 스펙트럼 영상의 대칭성과 샤플링

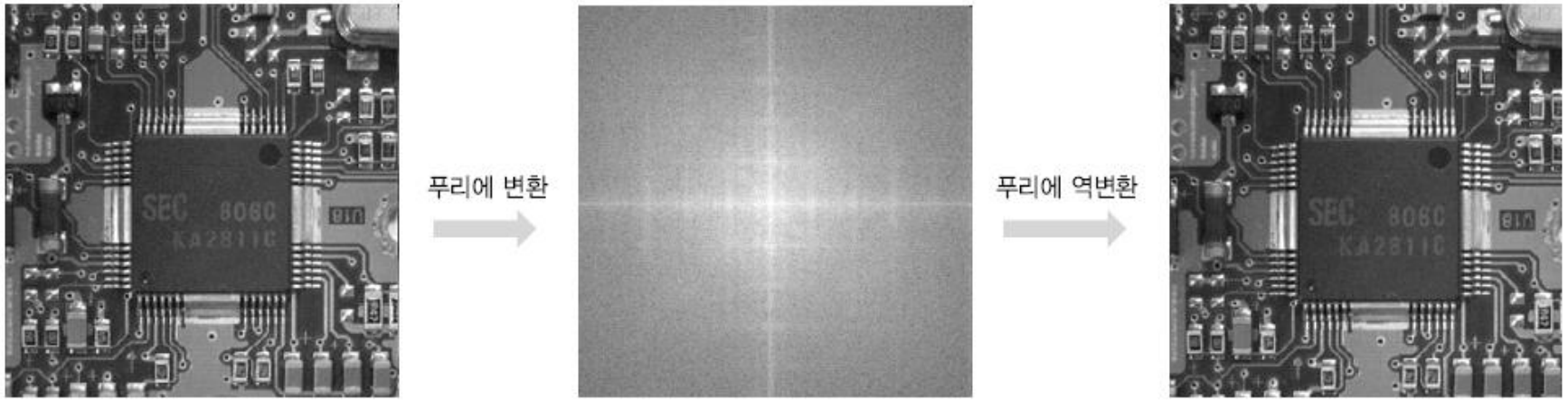


(a) 원본 영상 (b) 주파수 영상 (c) 샤플링 처리된 주파수 영상

[그림 13-12] FFT 처리된 영상과 셔플링 처리된 영상

역방향 고속 푸리에 변환

- 푸리에 변환은 가역 변환이므로 스펙트럼 영상에 푸리에 역변환을 적용하면 원래 공간 영역의 영상을 다시 얻을 수 있음.

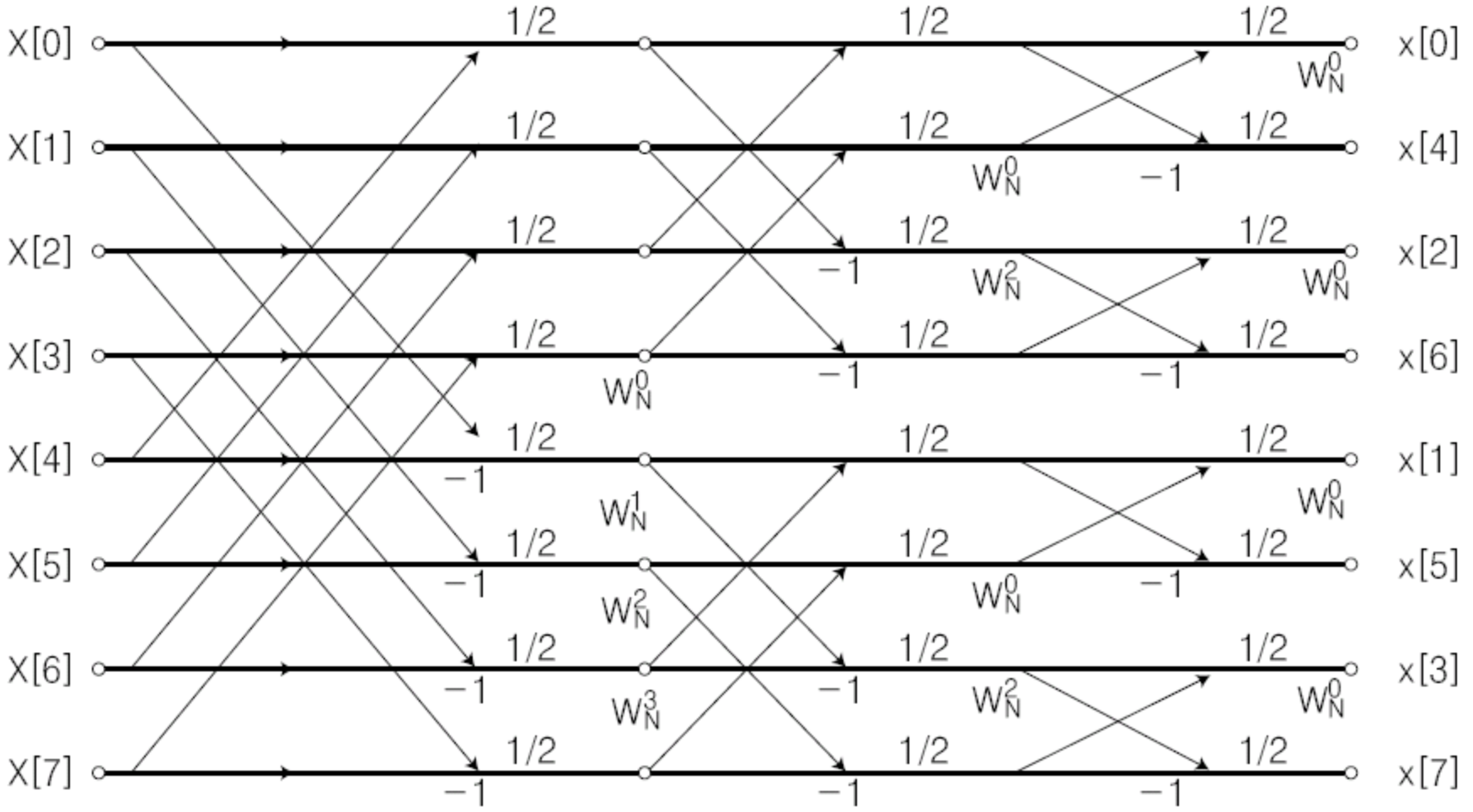


[그림 13-13] 푸리에 역변환의 결과

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k) W_N^{-nk}$$

- 역방향은 순방향에서과 **x(n)**과 **X(k)**의 위치를 바꾸고, $\mathbf{W_n}$을 $\mathbf{W_N^{-1}}$로 바꾸면 같게 됨.
- 스크램블링과 버터플라이 함수를 순방향과 같게 생성하고 이를 수행

역방향 고속 푸리에 변환(계속)



[그림 13-14] 역방향 고속 푸리에 변환(FFT) 프로그램의 수행

[실습하기 13-1] FFT 프로그램

① **ResourceView 창에서 [Menu]-[IDR_IMAGETYPE] 더블클릭 → 메뉴 추가**

ID	ID_FFT_2D
Caption	2차원 FFT

② **[MFC ClassWizard] 대화상자를 이용해 추가된 메뉴에서 2차원 FFT를 실행하는 함수 추가**

Class Name	Function Type	Function Name
View Class	void	OnFft2d
Doc Class	void	OnFft2d
Doc Class	void	OnFft1d(Complex *X, int N, int Log2N)
Doc Class	void	OnShuffle(Complex *X, int N, int Log2N);
Doc Class	void	OnButterfly(Complex *X, int N, int Log2N, int mode);
Doc Class	int	OnReverseBitOrder(int index, int Log2N);

[실습하기 13-1] FFT 프로그램

③ **Doc 클래스의 헤더 파일에 Complex 구조체 선언**

```
struct Complex{
    double Re; // 실수를 위한 변수
    double Im; // 허수를 위한 변수
};
```

④ **Doc 클래스에 FFT를 위한 변수 추가**

	Variable Type	Variable Name	Access
FFT 결과 저장을 위한 포인터	Complex	**m_FFT	Public

⑤ **Doc 클래스에 다음 프로그램 추가**

[실습하기 13-1] FFT 프로그램

❶ OnFft2d 함수 추가하기

```
void CImageProcessingDoc::OnFft2d()
{
   int i, j, row, col, Log2N, Num;
   Complex *Data;

   unsigned char **temp;
   double Value, Absol;

   Num = m_width;
   Log2N = 0;

   while(Num >= 2) // 영상의 너비 계산
   {
      Num >>= 1;
      Log2N++;
   }

   m_tempImage = Image2DMem(m_height, m_width); // 기억 장소 할당
   Data = new Complex [m_width];

   m_FFT = new Complex * [m_height];
   // 주파수 영역 변환 영상을 저장하기 위한 배열
   temp = new unsigned char * [m_height];

   for(i=0 ; i<m_height; i++){
      m_FFT[i] = new Complex [m_width];
      temp[i] = new unsigned char [m_width];
   }
```

[실습하기 13-1] FFT 프로그램

❶ OnFft2d 함수 추가하기(계속)

```
for(i=0 ; i<m_height ; i++){
     for(j=0 ; j<m_width ; j++){
        Data[j].Re = (double)m_InputImage[i*m_width+j];
        // 입력의 한 행을 복사, 실수 성분 값은 영상의 값
        Data[j].Im = 0.0; // 복소 성분 값은 0
     }
     OnFft1d(Data, m_width, Log2N); // 1차원 FFT
     for(j=0 ; j<m_width ; j++){ // 결과 저장
        m_FFT[i][j].Re = Data[j].Re;
        m_FFT[i][j].Im = Data[j].Im;
     }
  }

  Num = m_height;
  Log2N = 0;

  while(Num >= 2) // 영상의 높이 계산
  {
     Num >>= 1;
     Log2N++;
  }

  Data = new Complex [m_height];
```

[실습하기 13-1] FFT 프로그램

❶ OnFft2d 함수 추가하기 (계속)

```
    for(i=0 ; i<m_width ; i++){
      for(j=0 ; j<m_height ; j++){
         Data[j].Re = m_FFT[j][i].Re; // 영상의 한 열을 복사
         Data[j].Im = m_FFT[j][i].Im;
      }

      OnFft1d(Data, m_height, Log2N); // 1차원 FFT

      for(j=0 ; j<m_height ; j++){ // 결과 저장
         m_FFT[j][i].Re = Data[j].Re;
         m_FFT[j][i].Im = Data[j].Im;
      }
   }
// FFT 실행 결과를 영상으로 출력하기 위한 연산
   for(i=0 ; i<m_height ; i++){
      for(j=0 ; j<m_width ; j++){
         Value = sqrt((m_FFT[i][j].Re * m_FFT[i][j].Re)+
                  (m_FFT[i][j].Im * m_FFT[i][j].Im));
         Absol = 20 * log(Value);

         if(Absol > 255.0)
            Absol = 255.0;
         if(Absol < 0.0)
            Absol = 0.0;
         m_tempImage[i][j] = Absol;
      }
   }
```

[실습하기 13-1] FFT 프로그램

❶ **OnFft2d 함수 추가하기 (계속)**

```
    // 셔플링 과정 : 영상을 4등분하고 분할된 영상을 상하 대칭 및 좌우 대칭
  for(i=0 ; i<m_height; i += m_height / 2){
     for(int j=0 ; j<m_width; j += m_width / 2){
        for(row=0 ; row<m_height / 2 ; row++){
           for(col=0 ; col<m_width / 2 ; col++){
              temp[(m_height/2-1) - row + i][(m_width/2-1) - col + j]
                 = (unsigned char)m_tempImage[i + row][j + col];
           }
        }
     }
  }

  for(i = 0; i < m_height; i++){
     for(j = 0; j < m_width; j++){
        m_OutputImage[i*m_width+j] = temp[i][j];
     }
  }

  delete [] Data, **temp;
}
```

[실습하기 13-1] FFT 프로그램

❷ **OnFft1d 함수 추가하기**

```
void CImageProcessingDoc::OnFft1d(Complex *X, int N, int Log2N)
{
    // 1차원 fft를 위한 함수
    OnShuffle(X, N, Log2N); // 함수 호출
    OnButterfly(X, N, Log2N, 1); // 함수 호출
}
```

[실습하기 13-1] FFT 프로그램

❸ OnShuffle 함수 추가하기

```
void CImageProcessingDoc::OnShuffle(Complex *X, int N, int Log2N)
{
    // 입력 데이터의 순서를 바구기 위한 함수
    int i;
    Complex *temp;

    temp = new Complex [N];

    for (i=0 ; i<N ; i++) {
        temp[i].Re = X[OnReverseBitOrder(i, Log2N)].Re;
        temp[i].Im = X[OnReverseBitOrder(i, Log2N)].Im;
    }

    for (i=0 ; i<N ; i++) {
        X[i].Re = temp[i].Re;
        X[i].Im = temp[i].Im;
    }
    delete [] temp;
}
```

[실습하기 13-1] FFT 프로그램

❹ **OnButterfly 함수 추가하기**

```
void CImageProcessingDoc::OnButterfly(Complex *X, int N,
     int Log2N, int mode)
{
   // 나비(Butterfly) 구조를 위한 함수
   int i, j, k, m;

   int start;
   double Value;
   double PI = 3.14159265358979;

   Complex *Y, temp;

   Y = new Complex [N/2];

   for(i=0 ; i<Log2N ; i++){
      Value = pow(2, i+1);

      if(mode == 1){
         for(j=0 ; j<(int)(Value/2) ; j++){
            Y[j].Re = cos(j*2.0*PI / Value);
            Y[j].Im = -sin(j*2.0*PI / Value);
         }
      }
```

[실습하기 13-1] FFT 프로그램

❹ **OnButterfly 함수 추가하기 (계속)**

```
if(mode == 2){
        for(j=0 ; j<(int)(Value/2) ; j++){
           Y[j].Re = cos(j*2.0*PI / Value);
           Y[j].Im = sin(j*2.0*PI / Value);
        }
     }

     start = 0;

     for(k=0 ; k<N/(int)Value ; k++){
        for(j=start ; j<start+(int)(Value/2) ; j++){
           m = j + (int)(Value/2);
           temp.Re = Y[j-start].Re * X[m].Re
                     - Y[j-start].Im * X[m].Im;
           temp.Im = Y[j-start].Im * X[m].Re
                     + Y[j-start].Re * X[m].Im;

           X[m].Re = X[j].Re - temp.Re;
           X[m].Im = X[j].Im - temp.Im;

           X[j].Re = X[j].Re + temp.Re;
           X[j].Im = X[j].Im + temp.Im;
        }
        start = start + (int)Value;
     }
  }
```

[실습하기 13-1] FFT 프로그램

❹ **OnButterfly 함수 추가하기 (계속)**

```
  if(mode == 2){
      for(i=0 ; i<N ; i++){
         X[i].Re = X[i].Re / N;
         X[i].Im = X[i].Im / N;
      }
   }

   delete [] Y;
}
```

[실습하기 13-1] FFT 프로그램

❺ **OnReverseBitOrder 함수 추가하기**

```
int CImageProcessingDoc::OnReverseBitOrder(int index, int Log2N)
{
   int i, X, Y;

   Y = 0;

   for(i=0 ; i<Log2N ; i++){
      X = (index & (1<< i)) >> i;
      Y = (Y << 1) | X;
   }

   return Y;
}

         13-1 VIEW
void CImageProcessingView::OnIfft2d()
{
   CImageProcessingDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);

   pDoc->OnIfft2d();

   Invalidate(TRUE);
}
```

[실습하기 13-1] FFT 프로그램

⑥ View 클래스에 다음 프로그램 추가

```
void CImageProcessingView::OnNearest()
{
    CImageProcessingDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    pDoc->OnNearest();

    Invalidate(TRUE);
}
```

[실습하기 13-2] IFFT 프로그램

① **ResourceView 창에서 [Menu]-[IDR_IMAGETYPE] 더블클릭 → 메뉴 추가**

ID	ID_IFFT_2D
Caption	2차원 IFFT

② **[MFC ClassWizard] 대화상자를 이용해 추가된 메뉴에서 2차원 IFFT를 실행하는 함수 추가**

Class Name	Function Type	Function Name
View Class	void	OnIfft2d
Doc Class	void	OnIfft2d
Doc Class	void	OnIfft1d(Complex *X, int N, int Log2N)

③ **Doc 클래스에 FFT를 위한 변수 추가**

	Variable Type	Variable Name	Access
FFT 결과 저장을 위한 포인터	Complex	**m_IFFT	Public

④ **Doc 클래스에 다음 프로그램 추가**

[실습하기 13-2] IFFT 프로그램

❶ OnIfft2d 함수 추가하기

```
void CImageProcessingDoc::OnIfft2d()
{
    int i, j, Num, Log2N;
    Complex *Data;

    Num = m_width;
    Log2N = 0;
    while(Num >= 2) // 주파수 변환된 영상의 너비 계산
    {
        Num >>= 1;
        Log2N++;
    }

    Data = new Complex [m_height];
    m_IFFT = new Complex *[m_height]; // 역변환된 영상을 위한 배열

    for(i=0 ; i<m_height ; i++){
        m_IFFT[i] = new Complex [m_width];
    }
```

[실습하기 13-2] IFFT 프로그램

❶ OnIfft2d 함수 추가하기(계속)

```
for(i=0 ; i< m_height ; i++){
     for(j=0 ; j<m_width ; j++){ // 한 행을 복사
        Data[j].Re = m_FFT[i][j].Re;
        Data[j].Im = m_FFT[i][j].Im;
     }

     OnIfft1d(Data, m_width, Log2N); // 1차원 IFFT
     for(j=0 ; j<m_width ; j++){
        m_IFFT[i][j].Re = Data[j].Re; // 결과 저장
        m_IFFT[i][j].Im = Data[j].Im;
     }
  }

  Num = m_height;
  Log2N = 0;
  while(Num >= 2) // 주파수 변환된 영상의 높이 계산
  {
     Num >>= 1;
     Log2N++;
  }

  Data = new Complex [m_height];
```

[실습하기 13-2] IFFT 프로그램

❶ **OnIfft2d 함수 추가하기 (계속)**

```
   for(i=0 ; i<m_width ; i++){
       for(j=0 ; j<m_height ; j++){
          Data[j].Re = m_IFFT[j][i].Re; // 한 열을 복사
          Data[j].Im = m_IFFT[j][i].Im;
       }

       OnIfft1d(Data, m_height, Log2N); // 1차원 IFFT

       for(j=0 ; j<m_height ; j++){
          m_IFFT[j][i].Re = Data[j].Re; // 결과 저장
          m_IFFT[j][i].Im = Data[j].Im;
       }
    }

    for(i=0 ; i<m_width ; i++){
       for(j=0 ; j<m_height ; j++){
          m_OutputImage[i*m_width+j]
             = (unsigned char)m_IFFT[i][j].Re;
          // 결과 출력
       }
    }
    delete [] Data;
}
```

[실습하기 13-2] IFFT 프로그램

❷ **OnIfft1d 함수 추가하기**

```
void CImageProcessingDoc::OnIfft1d(Complex *X, int N, int Log2N)
{
    OnShuffle(X, N, Log2N);
    OnButterfly(X, N, Log2N, 2);
}

        13-2 VIEW
void CImageProcessingView::OnLpfFrequency()
{
    CImageProcessingDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    pDoc->OnLpfFrequency();

    Invalidate(TRUE);
}
```

[실습하기 13-2] IFFT 프로그램

④ View 클래스에 다음 프로그램 추가

```
void CImageProcessingView::OnNearest()
{
    CImageProcessingDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    pDoc->OnNearest();

    Invalidate(TRUE);
}
```

이산 코사인 변환(Discrete Cosine Transform: DCT)

- 영상을 압축하는 가장 효과적인 방법임이 검증
- 이산 코사인 변환은 푸리에 변환의 실수 부분의 코사인과 매우 비슷
- 기저 함수가 코사인 함수가 됨.
- 실수부만 다루므로 신호 처리를 효과적으로 수행할 수 있음.
- 1차원 이산 코사인 변환 쌍

$$F(u) = k(u)\sum_{x=0}^{N-1} f(x)\cos\left[\frac{(2x+1)u\pi}{2N}\right], \qquad u = 0,1,\ldots,N-1$$

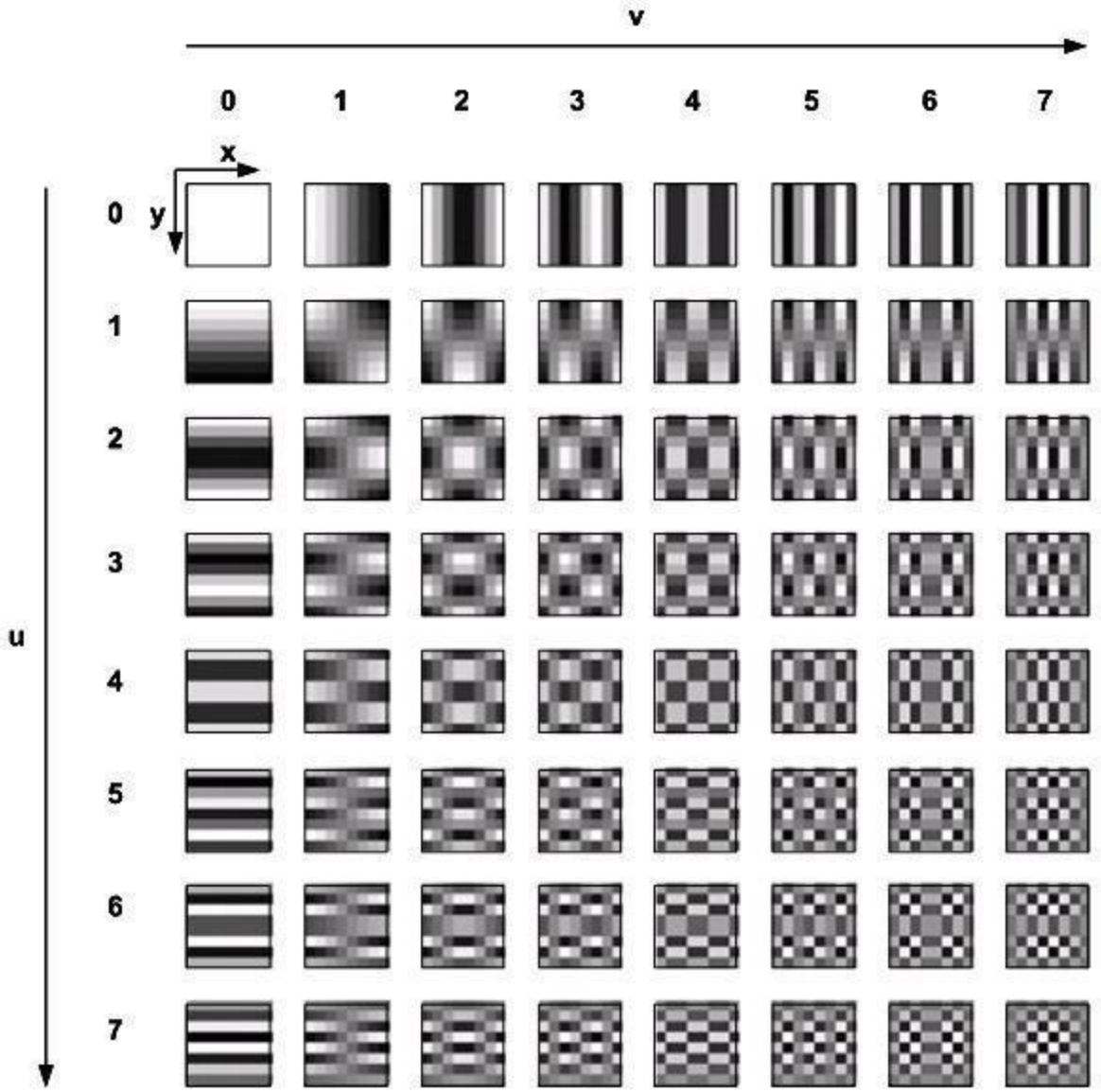
$$f(x) = \sum_{x=0}^{N-1} k(u)F(u)\cos\left[\frac{(2x+1)u\pi}{2N}\right], \qquad x = 0,1,\ldots,N-1$$

- 영상에 적용하는 2차원 이산 코사인 변환 쌍

$$F(u,v) = k(u)k(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

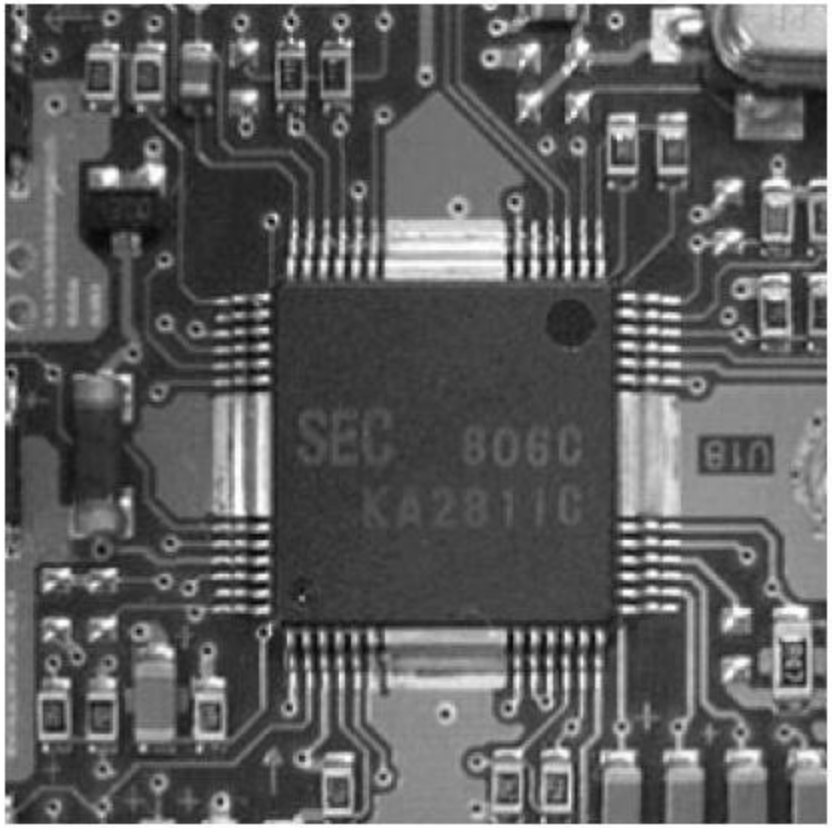
$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} k(u)k(v)F(u,v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

이산 코사인 변환(계속)

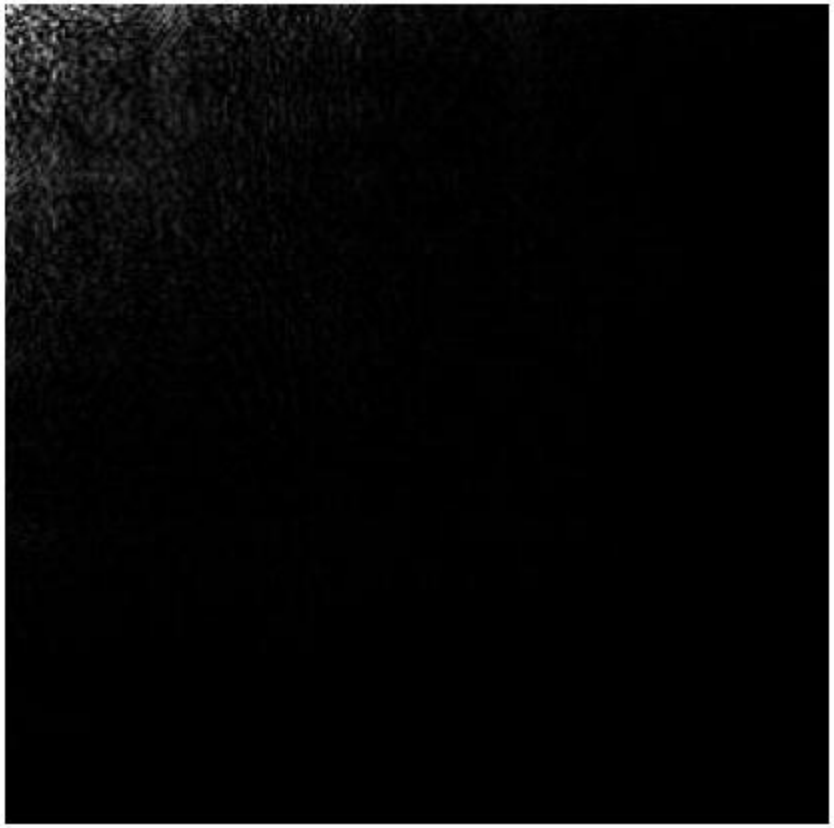


[그림 13-15] 이산 코사인 변환의 기저 함수

이산 코사인 변환(계속)



(a) 입력 영상

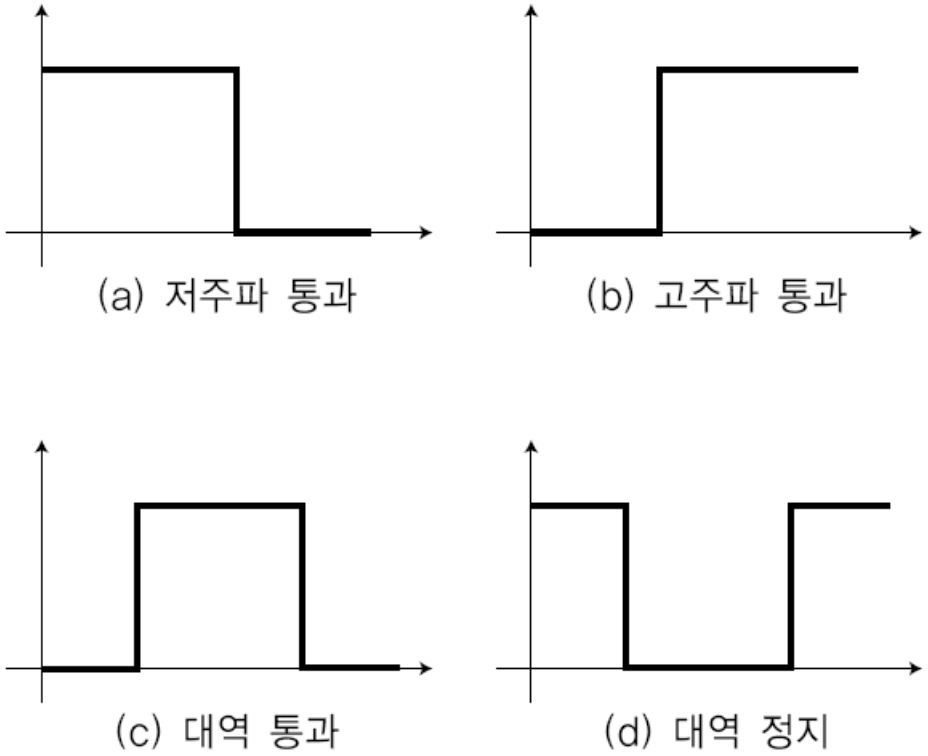


(b) 결과 영상

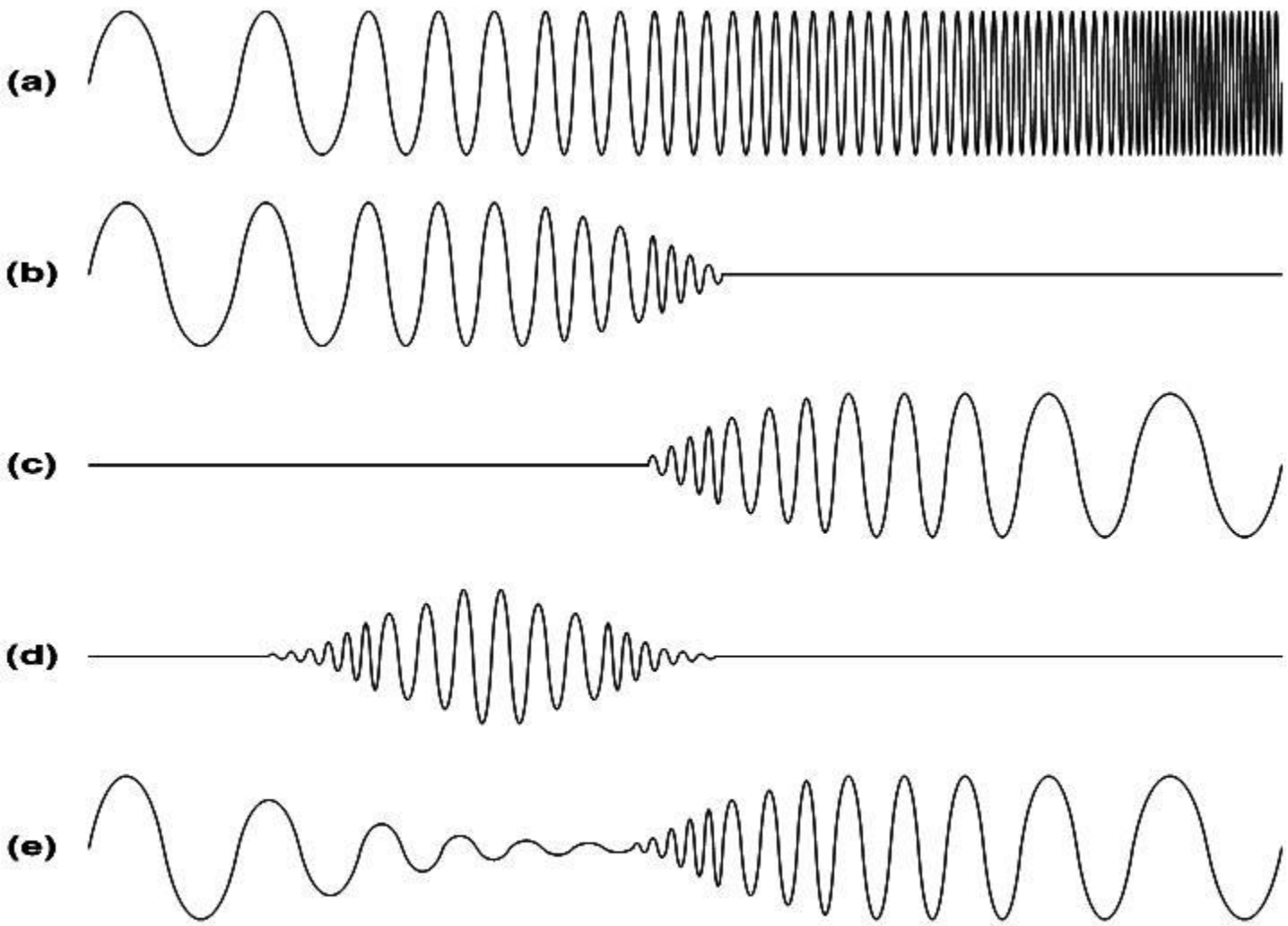
[그림 13-16] 이산 코사인 변환의 결과 영상

Section 03 주파수 영역에서의 필터링

- 영상의 푸리에 변환을 수행하는 목적 중 하나는 주파수 영역에서 필터링을 수행하기 위해서 임.
- 푸리에 변환 뒤 주파수 영역에서의 필터링은 영상에 포함된 주파수 성분을 파악하여 이를 토대로 영상에 포함된 주파수 성분을 필터링하는 것



[그림 13-17] 기본적인 필터의 종류



[그림 13-18] 기본 필터로 필터링한 결과

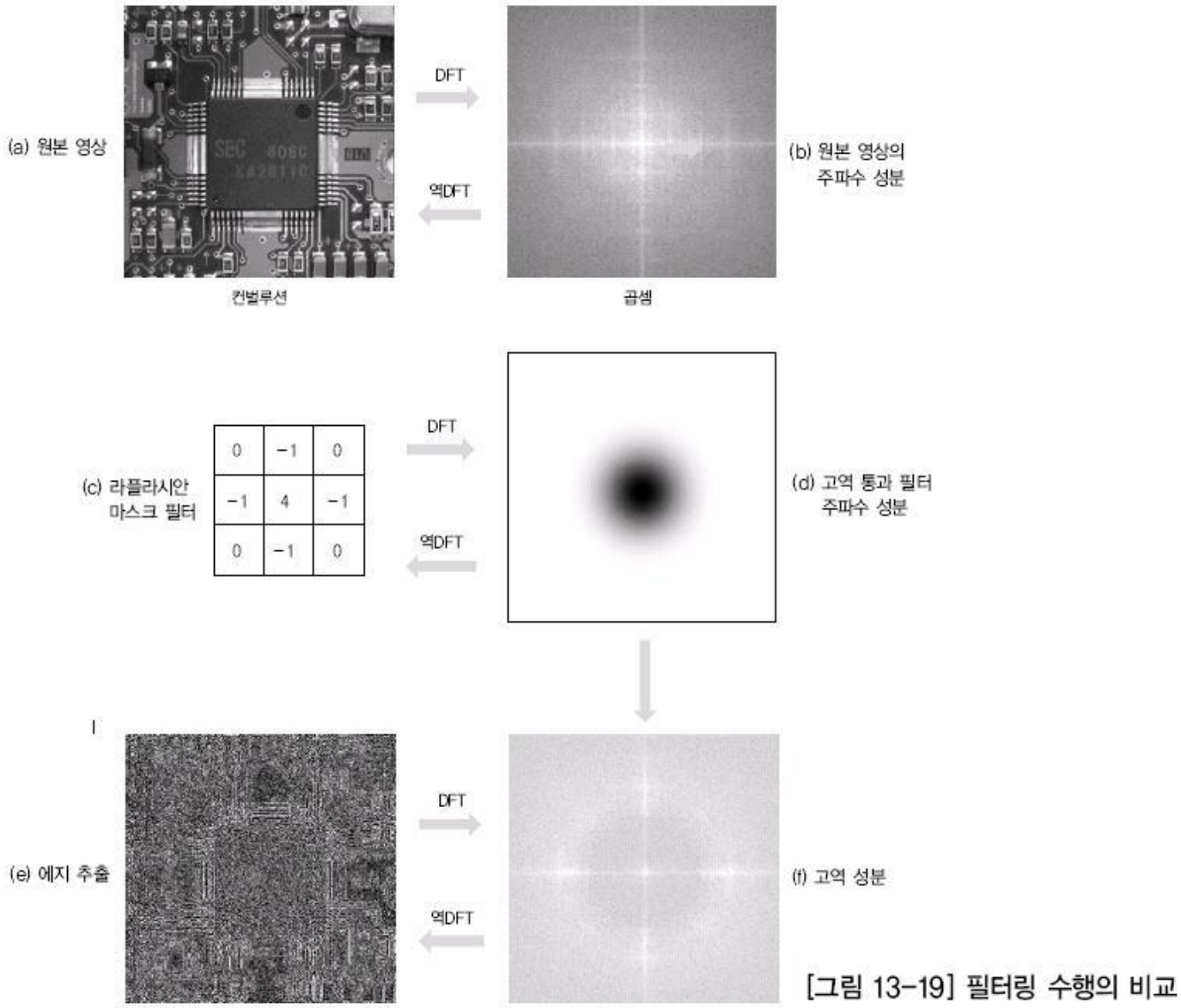
주파수 영역에서 필터링 수행 방법

- 주파수 영역에서 필터링의 수행은 공간 영역에서의 필터링보다 쉬움.
- 공간 영역에서의 필터링은 컨볼루션으로 수행하지만, 주파수 영역에서는 영상의 주파수 성분과 필터의 주파수 성분을 곱해서 해결
  → 컨벌루션 정리(Convolution Teorem)
- 컨벌루션 연산의 기호를 *로 표시

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

  - f(x, y)와 h(x, y)는 공간 영역에서의 영상 데이터와 필터 계수
  - F(u, v)와 H(u, v)는 주파수 영역에서 영상 주파수 데이터와 필터 계수의 주파수 데이터
  - u와 v는 x와 y 방향의 주파수 성분

주파수 영역에서 필터링 수행 방법(계속)



[그림 13-19] 필터링 수행의 비교

주파수 영역에서 필터링 수행 방법(계속)

- 필터에 주파수 영역의 마스크가 주어진다면 주파수 영역에서 필터링은 다음 순서를 따름.

  ① 영상의 푸리에 변환을 구한다.

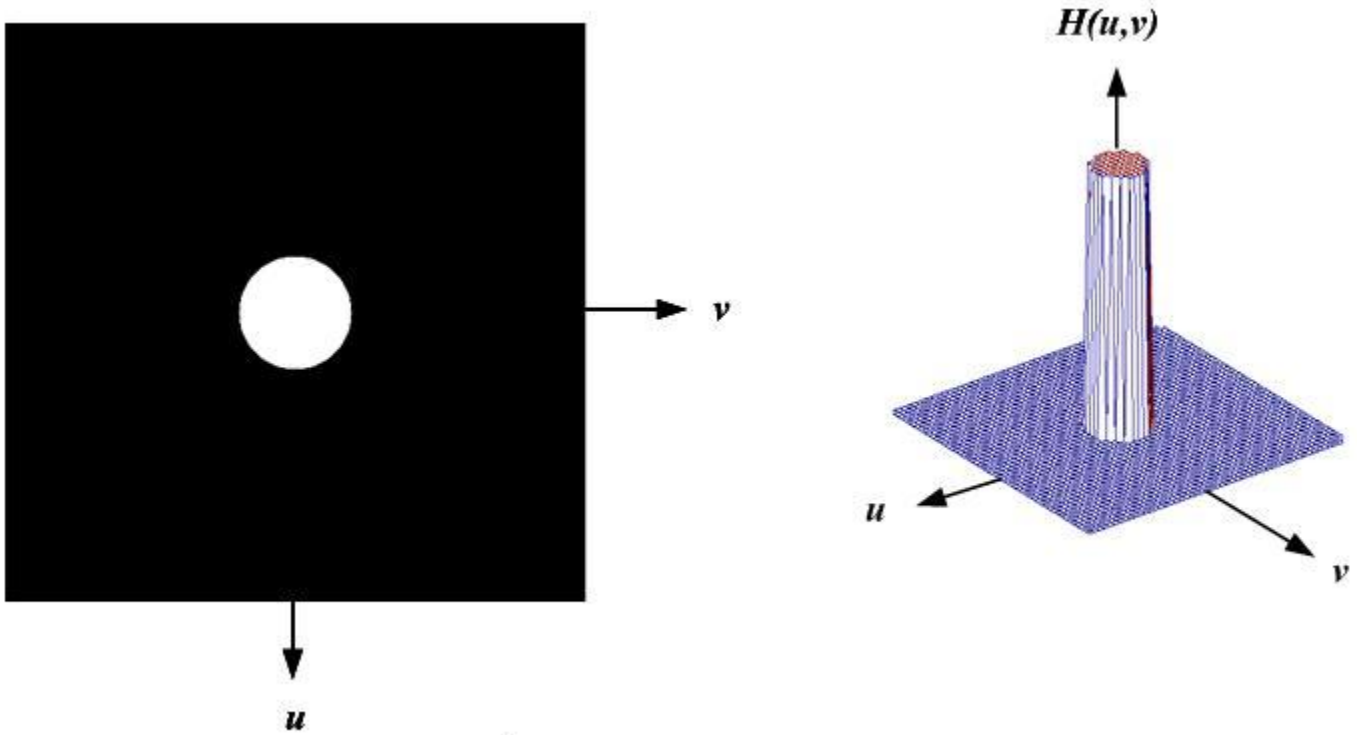
  ② 푸리에 변환된 영상과 필터 마스크를 곱한다.

  ③ ②의 결과에 역푸리에 변환을 구한다.

주파수 영역에서 저주파와 고주파 필터링

- 주파수 영역에서 필터링을 수행하려면, 먼저 주파수 영역의 필터 마스크를 만들어야 함.
  - 첫 번째 생성 방법: 공간 영역 필터 마스크를 주파수 영역 필터 마스크로 변환하는 것. 저주파와 고주파 필터 마스크를 푸리에 변환으로 얻을 수 있음(12장에서 소개)
  - 두 번째 생성 방법 : 주파수 영역에서 직접 마스크를 계산하여 얻는 것
- 이상적인 저주파 통과 필터의 생성
  - 고주파 성분을 감쇄시켜 영상을 흐릿하게 만드는 이상적인 저주파 통과 필터(Ideal Low Pass Filter)는 원점에서 어느 거리 내의 저주파 성분은 1을 곱해 통과시키고, 거리 밖의 고주파 성분은 0을 곱해 차단하도록 주파수 영역을 설계
  - 이상적인 저주파 통과 필터 수식
  - $$H(u,v) = \begin{cases} 1, & r(u,v) \le r_0 \\ 0, & r(u,v) > r_0 \end{cases}$$
    - r_0는 필터의 반경이며, 차단 주파수라고 함.

주파수 영역에서 저주파와 고주파 필터링(계속)



[그림 13-20] 이상적인 저주파 통과 필터의 특성

[실습하기 13-3] 주파수 영역에서 저주파 통과 필터링 프로그램

① **ResourceView 창에서 [Menu]-[IDR_IMAGETYPE] 더블클릭 → 메뉴 추가**

ID	ID_LPF_FREQUENCY
Caption	저주파 필터링(주파수영역)

② **[MFC ClassWizard] 대화상자를 이용해 추가된 메뉴에서 저주파 필터링(주파수 영역)을 실행하는 함수 추가**

Class Name	Function Type	Function Name
View Class	void	OnLpfFrequency
Doc Class	void	OnLpfFrequency

③ **Doc 클래스에 다음 프로그램 추가**

[실습하기 13-3] 주파수 영역에서 저주파 통과 필터링 프로그램

```
void CImageProcessingDoc::OnLpfFrequency()
{
   int i, j, x, y;
   double temp, D, N;
   D = 32.0;
   N = 4.0;

   OnFft2d(); // 주파수 변환

   // 주파수 변환된 값에서 고주파 성분 제거
   for(i=0 ; i<m_height ; i++) {
      for(j=0 ; j<m_width ; j++){
         x = i;
         y = j;
         if(x > m_height / 2)
            x = x - m_height;
         if(y > m_width / 2)
            y = y - m_width;

         temp = 1.0 / (1.0 + pow(sqrt((double)
               (x * x + y * y)) / D, 2*N));

         m_FFT[i][j].Re = m_FFT[i][j].Re * temp;
         m_FFT[i][j].Im = m_FFT[i][j].Im * temp;
      }
   }

   OnIfft2d(); // 주파수 역변환
}
```

[실습하기 13-3] 주파수 영역에서 저주파 통과 필터링 프로그램

④ View 클래스에 다음 프로그램 추가

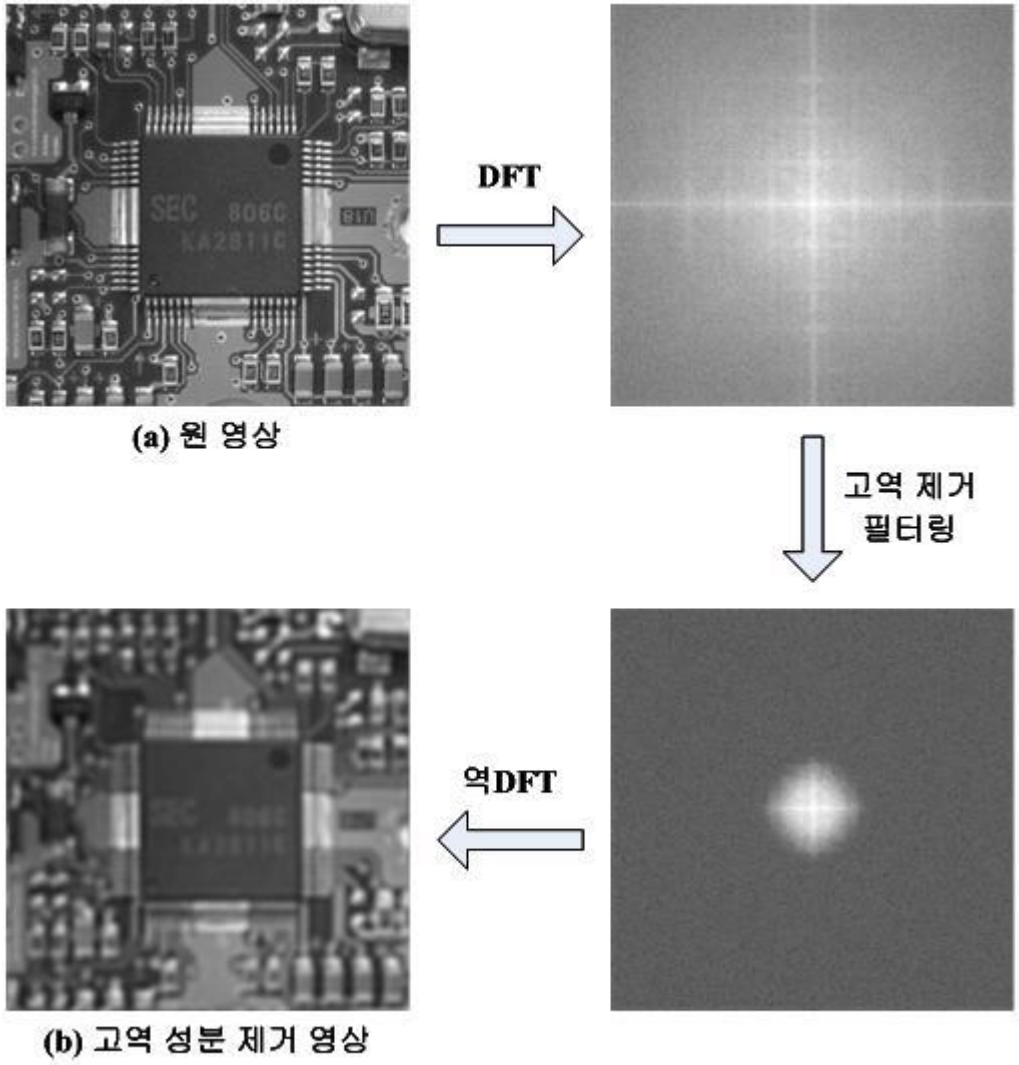
```
void CImageProcessingView::OnLpfFrequency()
{
    CImageProcessingDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    pDoc->OnLpfFrequency();

    Invalidate(TRUE);
}
```

[실습하기 13-3] 주파수 영역에서 저주파 통과 필터링 프로그램

⑤ 프로그램 실행 결과 영상

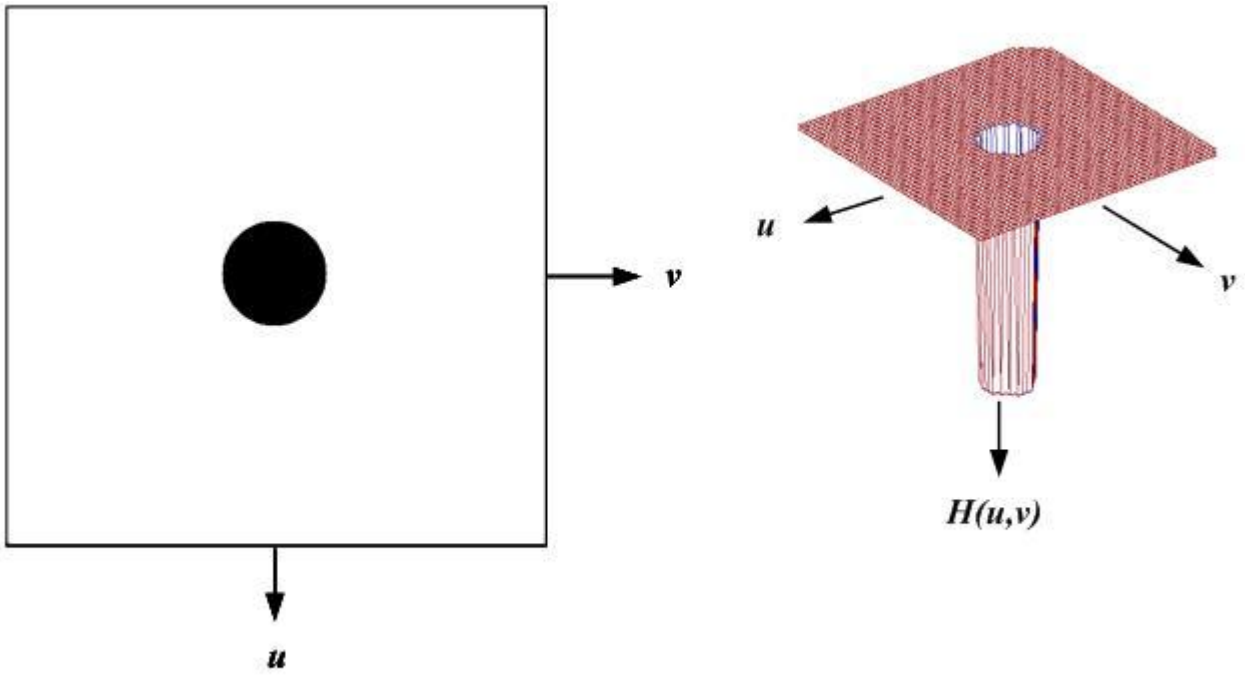


이상적인 저주파 통과 필터링의 결과 영상

이상적인 고주파 통과 필터의 생성

- 저주파 성분을 감쇄시켜 영상을 선명하게 만드는 이상적인 고주파 필터 **(Ideal High Pass Filter)**의 생성은 저주파 통과 필터에서 다음 수식으로 얻을 수 있음.

$$H_{high}(u,v) = 1 - H_{low}(u,v)$$



[그림 13-21] 이상적인 저주파 통과 필터의 특성

[실습하기 13-4] 주파수 영역에서 고주파 통과 필터링 프로그램

① **ResourceView 창에서 [Menu]-[IDR_IMAGETYPE] 더블클릭 → 메뉴 추가**

ID	ID_HPF_FREQUENCY
Caption	고주파 필터링(주파수영역)

② **[MFC ClassWizard] 대화상자를 이용해 추가된 메뉴에서 인접한 이웃 화소 보간법을 실행하는 함수 추가**

Class Name	Function Type	Function Name
View Class	void	OnHpfFrequency
Doc Class	void	OnHpfFrequency

③ **Doc 클래스에 다음 프로그램 추가**

[실습하기 13-4] 주파수 영역에서 고주파 통과 필터링 프로그램

```
void CImageProcessingDoc::OnHpfFrequency()
{
   int i, j, x, y;
   double temp, D, N;
   D = 128.0;
   N = 4.0;

   OnFft2d(); // 주파수 변환
   // 주파수 변환된 값에서 저주파 성분 제거
   for(i=0 ; i<m_height ; i++) {
      for(j=0 ; j<m_width ; j++){
         x = i;
         y = j;

         if(x > m_height / 2)
            x = x - m_height;
         if(y > m_width / 2)
            y = y - m_width;

         temp = 1.0 / (1.0 + pow(D / sqrt((double)
                (x * x + y * y)) , 2*N));

         m_FFT[i][j].Re = m_FFT[i][j].Re * temp;
         m_FFT[i][j].Im = m_FFT[i][j].Im * temp;
      }
   }
   OnIfft2d(); // 주파수 역변환
}
```

[실습하기 13-4] 주파수 영역에서 고주파 통과 필터링 프로그램

④ View 클래스에 다음 프로그램 추가

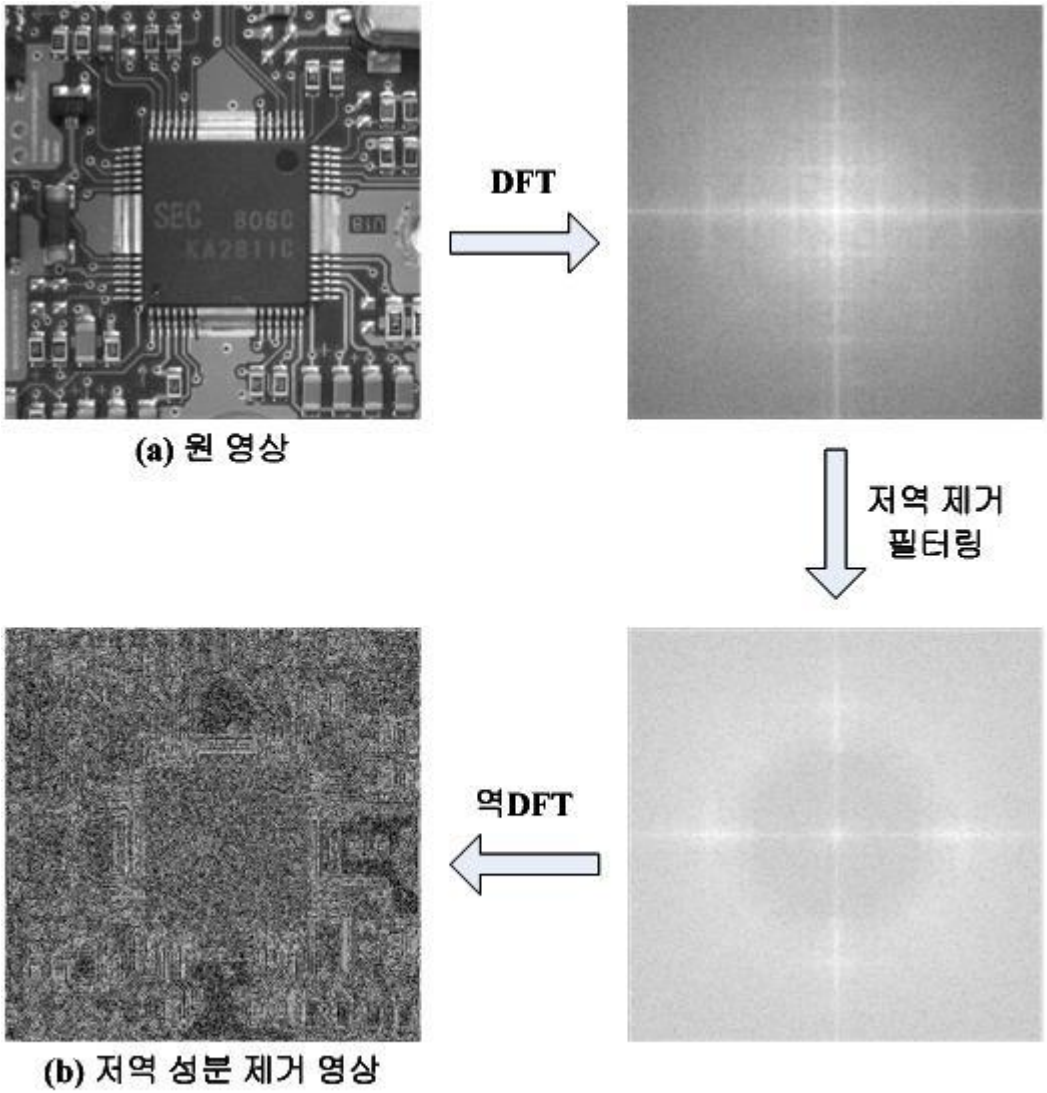
```
void CImageProcessingView::OnHpfFrequency()
{
    CImageProcessingDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    pDoc->OnHpfFrequency();

    Invalidate(TRUE);
}
```

[실습하기 13-4] 주파수 영역에서 고주파 통과 필터링 프로그램

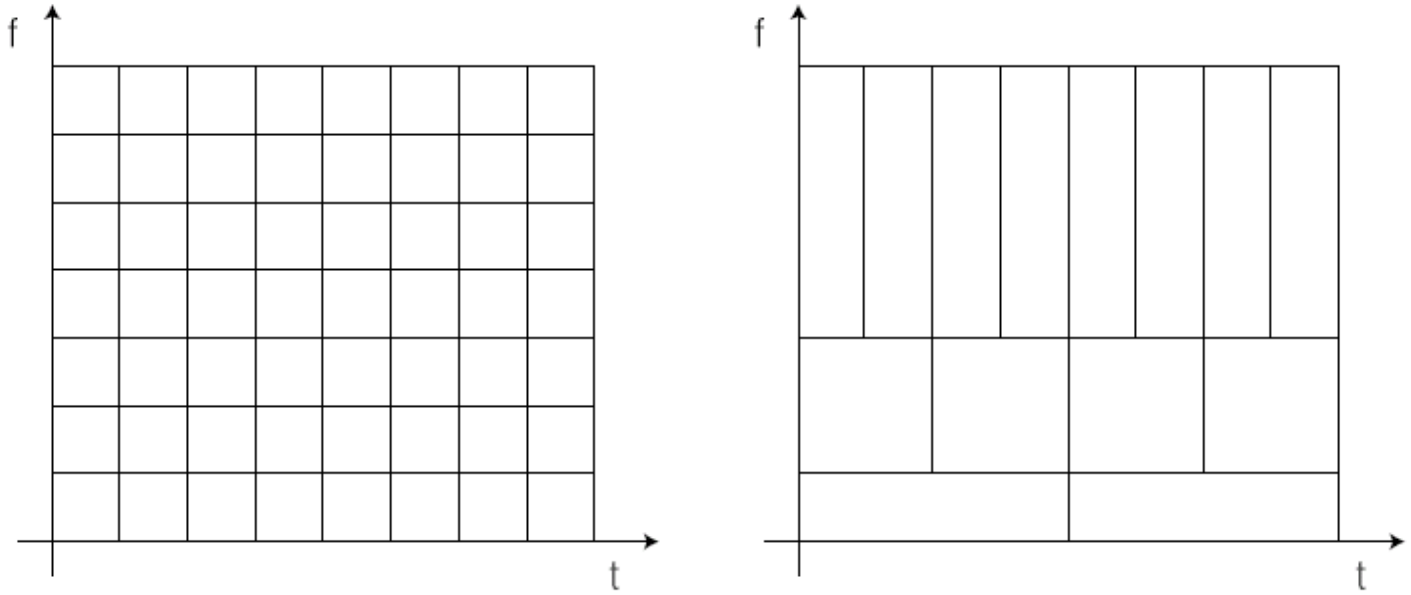
⑤ 프로그램 실행 결과 영상



이상적인 고주파 통과 필터링의 결과 영상

웨이브렛 이론

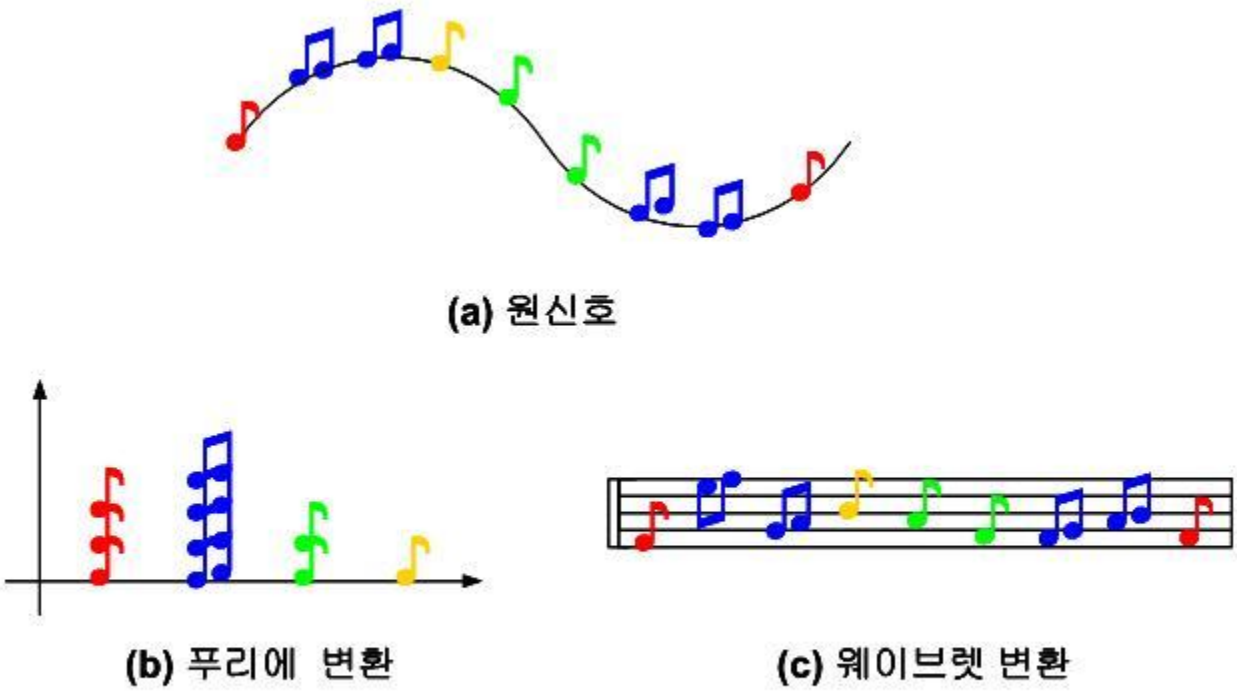
- 웨이브렛은 기본 함수로 **sine, cosine** 함수 외에 웨이브렛 모함수를 사용
- 각 주파수 영역에 따라 변화하는 다양한 기저 함수를 생성하여 사용
- 시간-주파수에 국부적인 성질이 있음.
- 푸리에 변환에서는 시간과 주파수 정보를 동시에 파악할 수 없지만, 웨이브렛에서는 이 둘을 동시에 파악 가능



(a) 푸리에 기저 함수 (b) 웨이브렛 기저 함수

[그림 13-22] 푸리에 변환과 웨이브렛 변환의 기저 함수 비교

웨이브렛 이론(계속)



[그림 13-23] 푸리에 변환과 웨이브렛 변환의 특징 비교

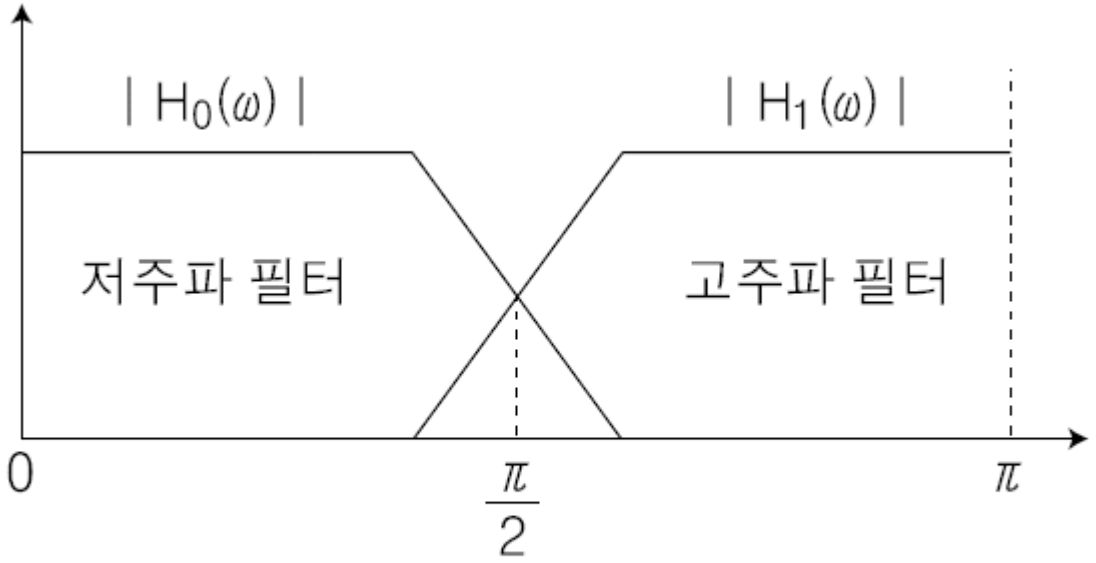
필터 뱅크를 이용한 이산 웨이브렛 변환 수행

- 이산 웨이브렛 변환은 저주파 통과 필터와 고주파 통과 필터로 구성된 필터 뱅크로 수행
- 사용되는 필터는 특수하게 설계된 것으로, 직교 특성, 선형 특성, 고주파와 저주파 부분을 정확하게 분할하는 특성이 있음.
- 더비쉬에(Daubechies)가 제안한 계수의 길이가 4인 웨이브렛 필터 계수

저주파 통과 필터의 계수 : $h_0[n]=\frac{1}{4\sqrt{2}}[1+\sqrt{3}\quad 3+\sqrt{3}\quad 3-\sqrt{3}\quad 1-\sqrt{3}]$

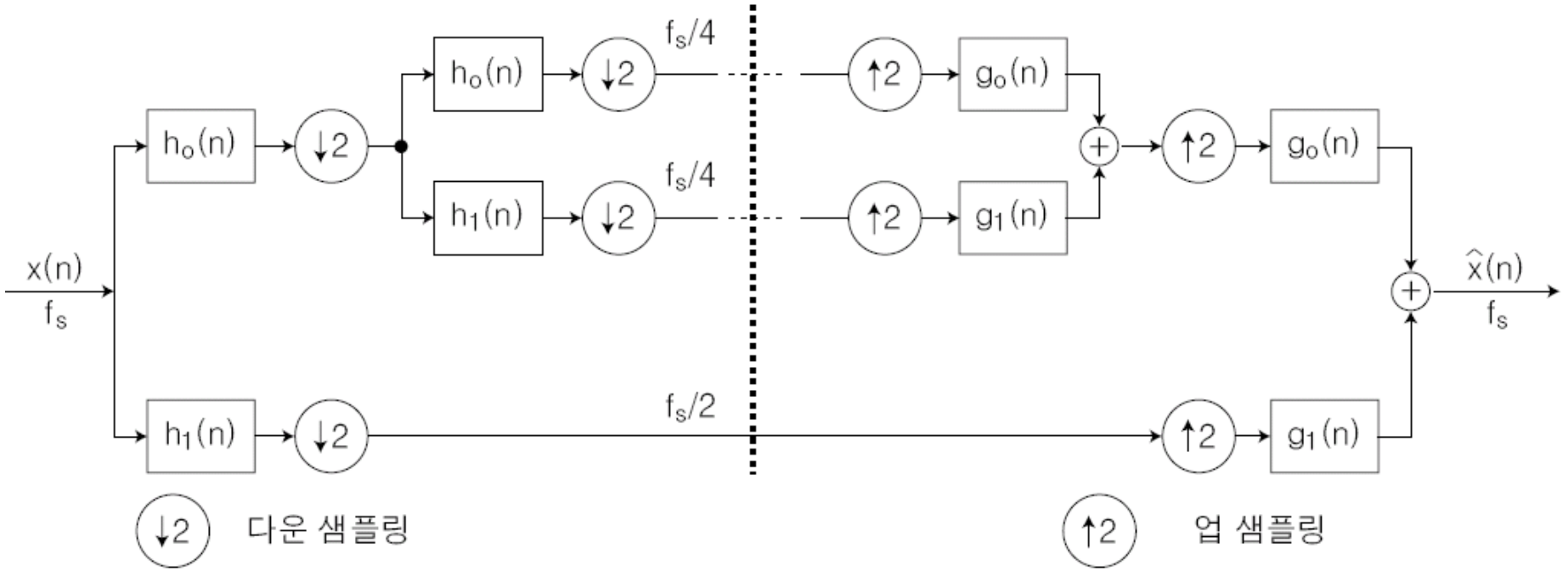
고주파 통과 필터의 계수 : $h_1[n]=\frac{1}{4\sqrt{2}}[1-\sqrt{3}\quad \sqrt{3}-3\quad 3+\sqrt{3}\quad -1-\sqrt{3}]$

필터 뱅크를 이용한 이산 웨이브렛 변환 수행(계속)



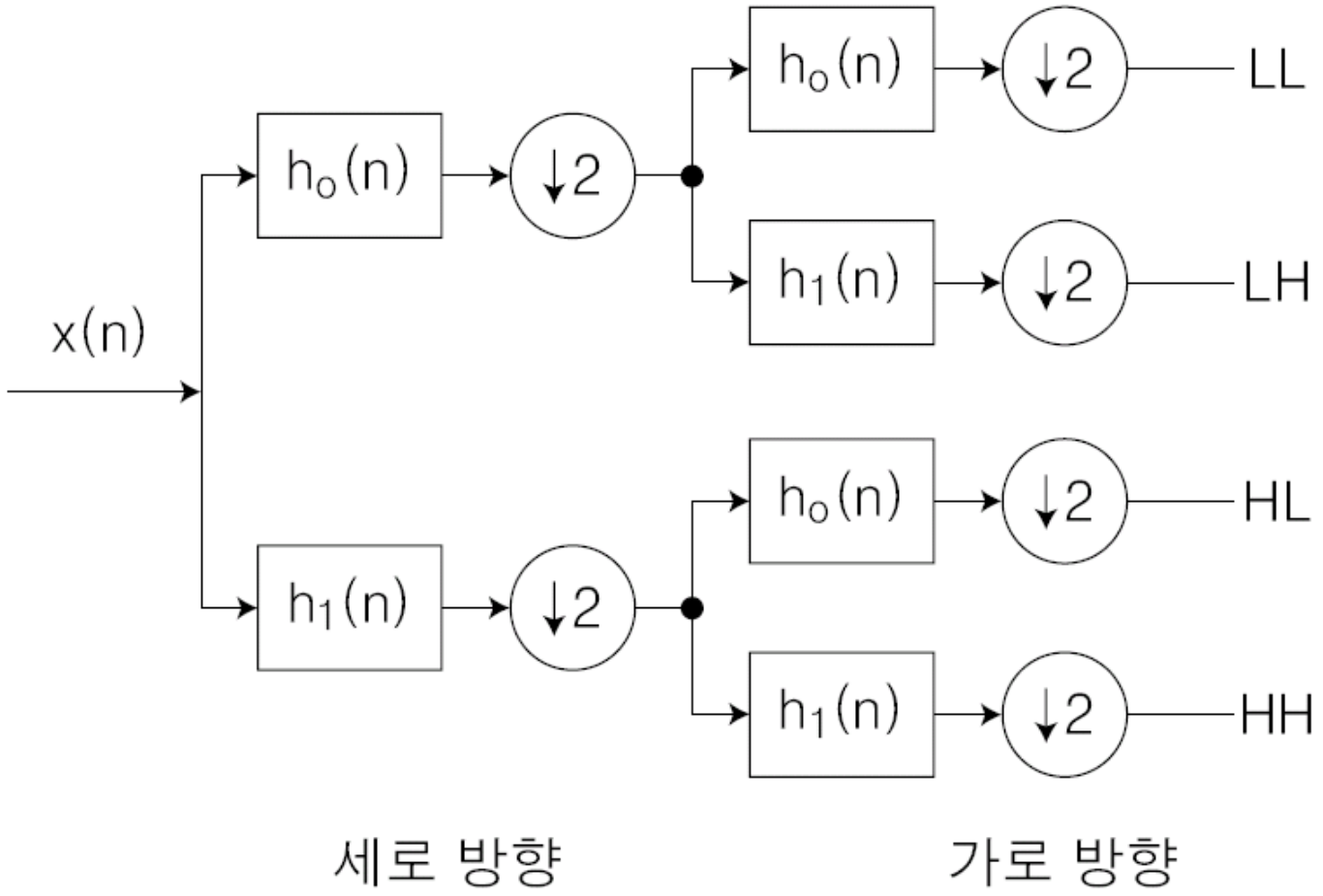
[그림 13-24] 웨이브렛 필터의 주파수 응답

필터 뱅크를 이용한 이산 웨이브렛 변환 수행(계속)



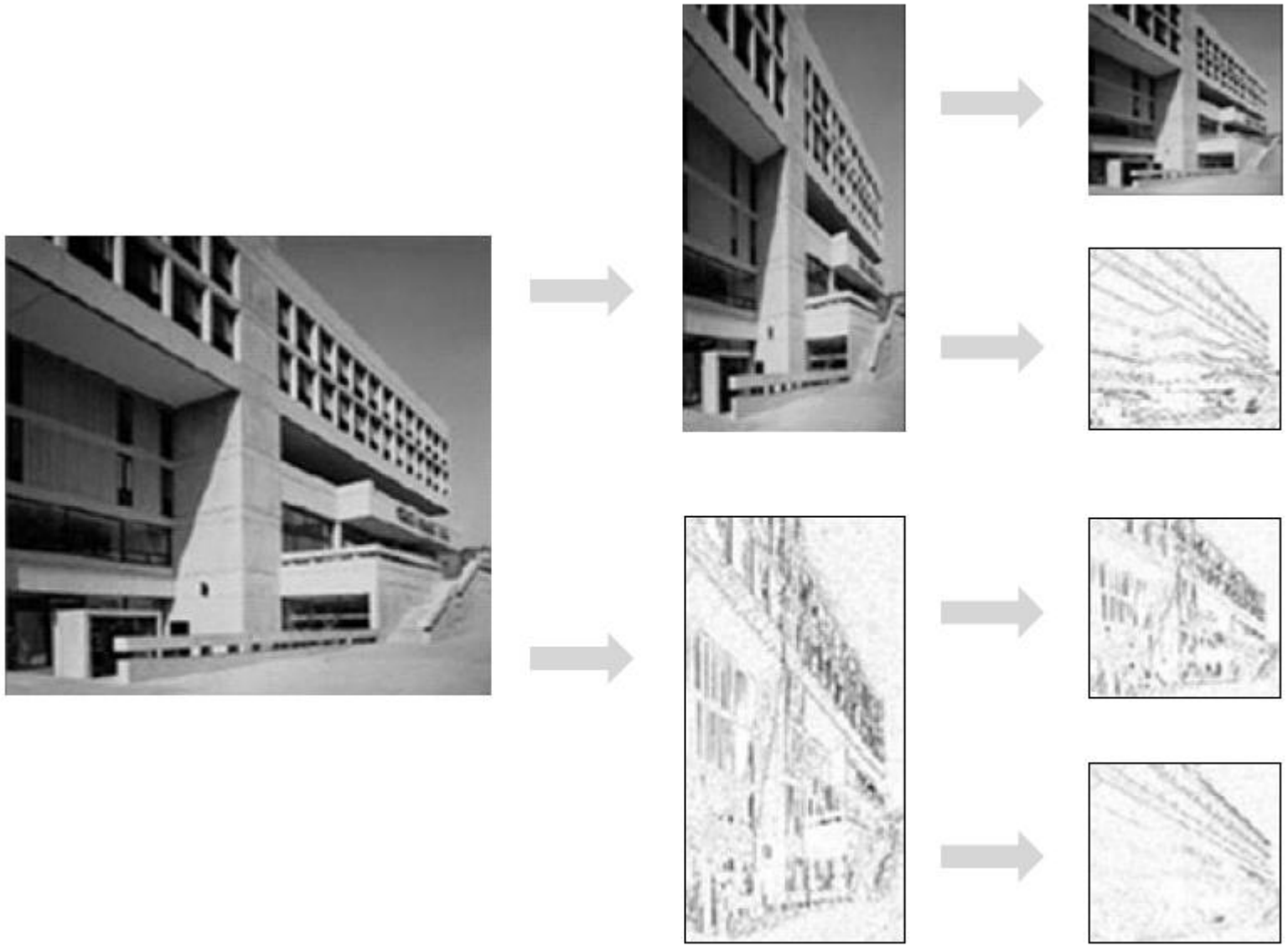
[그림 13-25] 필터 뱅크를 이용한 웨이브렛 변환과 역변환

필터 뱅크를 이용한 이산 웨이브렛 변환 수행(계속)



[그림 13-26] 2차원 웨이브렛 변환

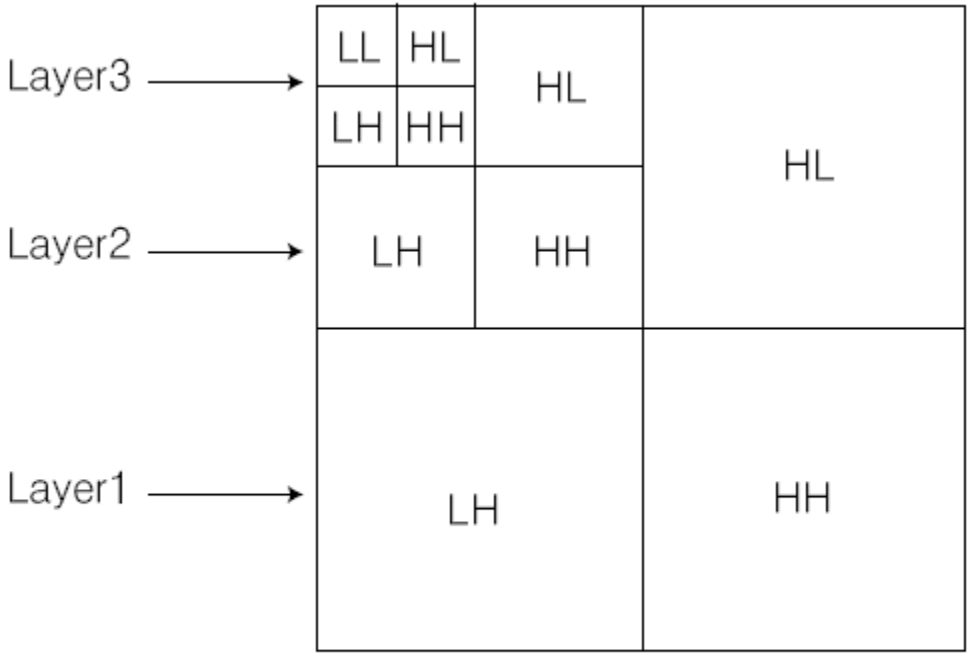
필터 뱅크를 이용한 이산 웨이브렛 변환 수행(계속)



[그림 13-27] 실제 영상에서의 2차원 웨이브렛 변환 결과 영상

필터 뱅크를 이용한 이산 웨이브렛 변환 수행(계속)

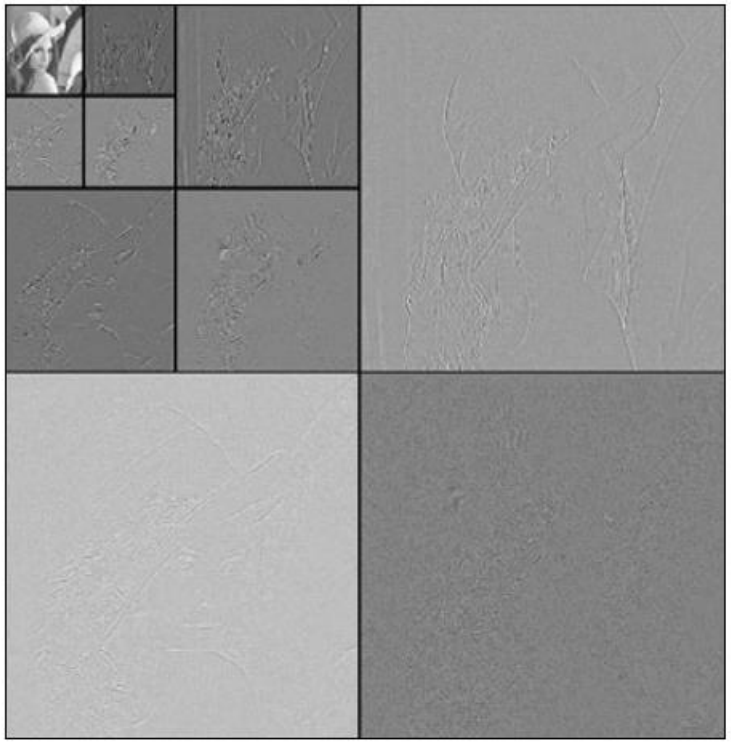
- 웨이브렛 변환의 결과로 형성된 부영상의 대역별 배치를 옥타브 나무 **(Octave Tree Structure)** 분할이라고 함.



[그림 13-28] 옥타브 나무 분할 구조

필터 뱅크를 이용한 이산 웨이브렛 변환 수행(계속)

- 실제 영상을 삼 단계 옥타브 나무 분할 구조로 배치한 것
- 여러 가지 주파 성분이 있고, 여러 부영상의 크기 합은 원래 영상과 같음.



[그림 13-29] 실제 영상에서의 옥타브 나무 분할 구조

요약

- 영상 변환
  - 영상 데이터를 다른 형태의 데이터로 변환하는 것으로, 데이터를 바라보는 관점을 변경하여 새로운 정보를 얻는 것
  - 영상의 개선, 복원, 압축, 해석 등 다양한 영상처리 작업을 공간에서 처리하는 것보다 더 쉽고 효율적으로 수행할 수 있게 함.
- 주파수
  - 영상에서 밝기의 변화 정도를 나타내는 것으로, 화소 값의 변화율을 뜻함.
  - 밝기가 얼마나 빨리 변화하는가에 따라서 고주파와 저주파로 분류
- 영상에서 높은 주파수 성분을 낮추면
  - 섬세한 부분이 사라지고, 부드럽고 엉성한 영상으로 변함.
- 낮은 주파수 성분을 낮추면
  - 엉성한 부분이 사라지면서 섬세한 부분에 해당하는 경계가 강조됨.
- 주파수 변환
  - 공간 영역 형태의 영상을 주파수 영역 형태의 기본 주파수로 분리하는 것
  - 정규적인 변환이 성립하려면 역변환도 성립되어야 함.

요약

- 푸리에 변환
  - 주파수 영역으로 변환하는 가장 일반적인 방법
- 고속의 푸리에 변환(FFT)
  - 이산 푸리에 변환 공식에서 반복 계산을 제거하면 변환을 빠르게 수행할 수 있는 장점이 있음.
- 1차원 FFT는 두 단계로 구현됨
  - 1단계: 스크램블링 단계

    재귀적인 DFT 계산 주기와 맞추려고 데이터를 적절히 재배치함.
  - 2단계 : 버터플라이 함수 적용 단계

    데이터를 점의 집합으로 나눠 이웃한 점의 DFT 변환을 수행함.
- 스펙트럼 영상
  - 디지털 영상이 이산 푸리에 변환으로 주파수 영역 영상으로 변환되는 것
  - 스펙트럼의 상용 대수식을 이용하여 스펙트럼 영상을 생성함.

Thank you