패턴 인식

강미선

개요

- 픽셀 기반 처리
  - 픽셀의 원래 값이나 위치에 기반한 픽셀 값 변경
  - 다른 픽셀의 영향을 받지 않음

- 픽셀 기반 처리기법들
  - 산술 연산, 히스토그램 평활화, 명암대비 스트레칭, 이진화

산술연산

- 화소에 일정한 값을 더하거나 빼거나 나누거나 곱하는 연산

- 덧셈 연산 및 뺄셈 연산
  - 영상의 밝기 조절

- 곱셈 연산 및 나눗셈 연산
  - 영상의 명암 대비를 조절

산술연산 예

입력영상

영상 + 40

영상 - 40

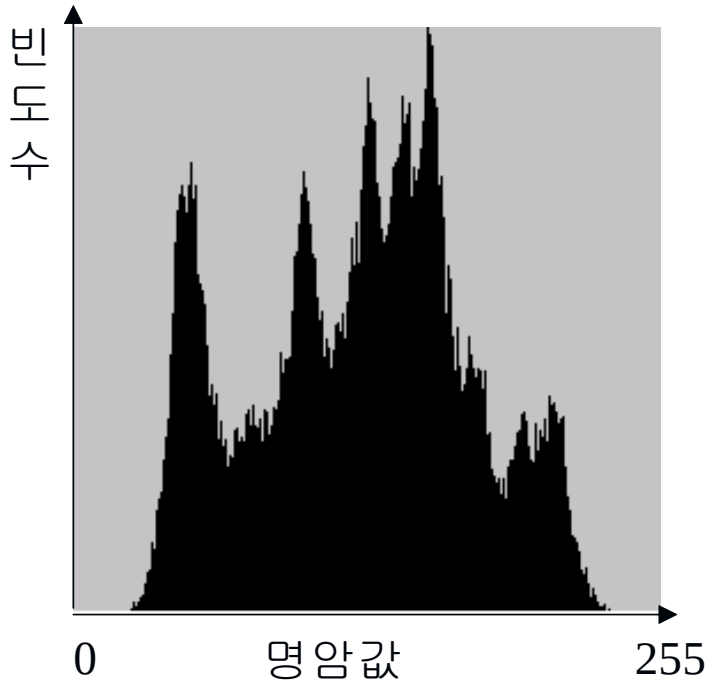
명암대비
증가

영상 * 1.2

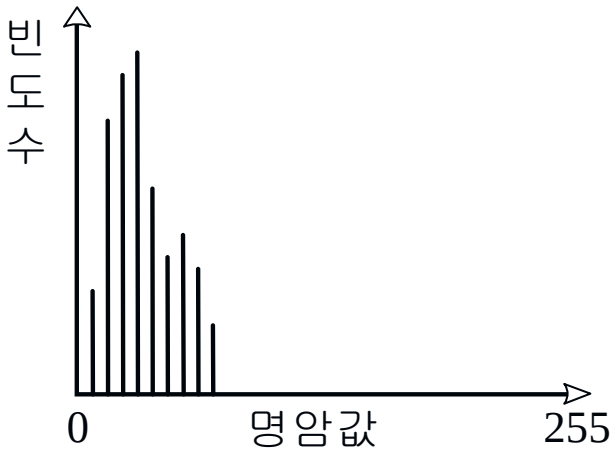
영상 / 1.2

히스토그램

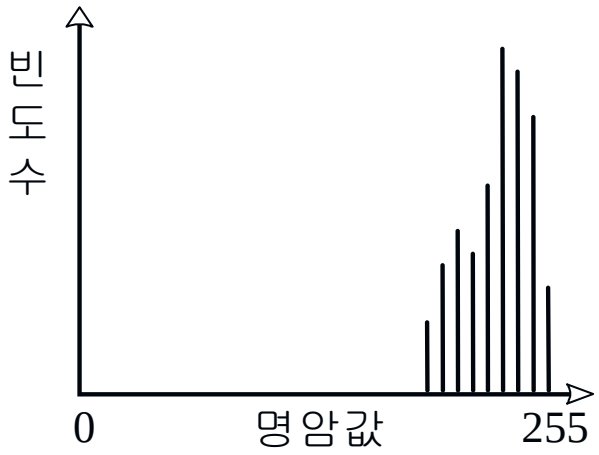
- 화소가 가진 명암 값에 대한 막대 그래프



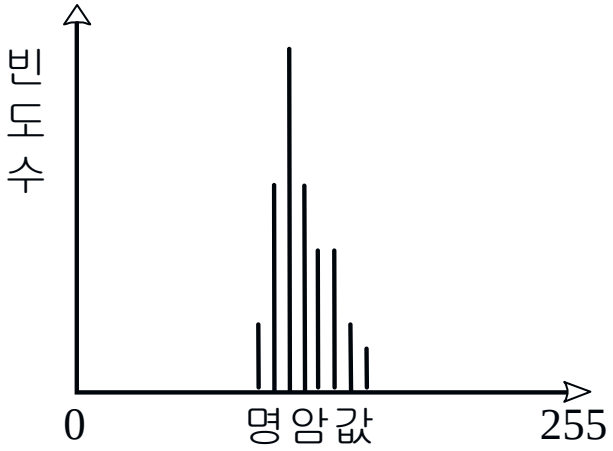
히스토그램



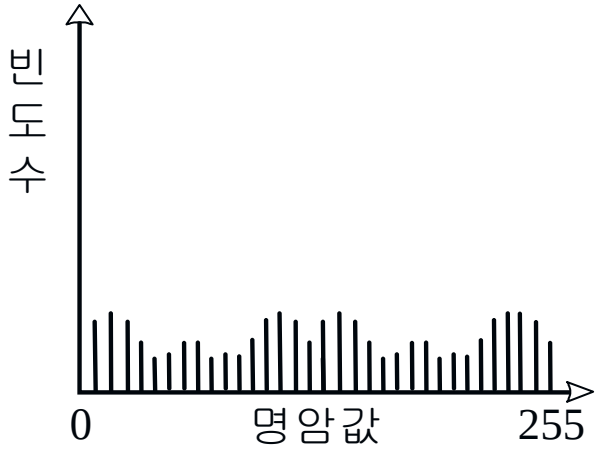
(a) 어두운 영상



(b) 밝은 영상



(c) 낮은 명암대비 영상



(d) 높은 명암대비 영상

히스토그램 평활화(equalization)

- 기존 영상의 명암 값 분포를 재분배하여 일정한 분포를 가진 히스토그램을 생성

- 히스토그램 평활화의 네 단계

1. 입력영상의 히스토그램 생성
   - 명암값 j 의 빈도수 $hist[j]$ 를 계산

2. 각 명암값 i 에 대하여 0부터 i 까지의 빈도수의 누적 값을 계산

$$sum[i] = \sum_{j=0}^{i} hist[j]$$

히스토그램 평활화(equalization)

$$n[i] = sum[i] \times \frac{1}{N} \times 255 \quad (N = \text{전체 픽셀 수})$$

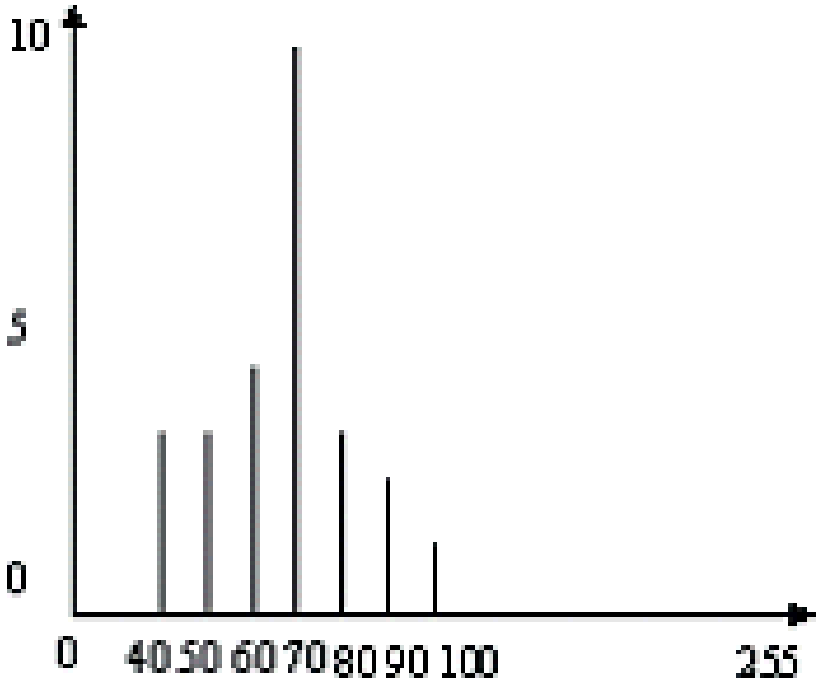
3. 단계 2에서 구한 누적값을 **정규화**

4. 입력영상에서 픽셀 값 i를 정규화된 값 n[i]로 변환하여 결과 영상 생성

히스토그램 평활화 예

40	40	50	60	70
40	50	60	70	70
50	60	70	70	80
60	70	70	80	90
70	70	80	90	100

원영상



히스토그램

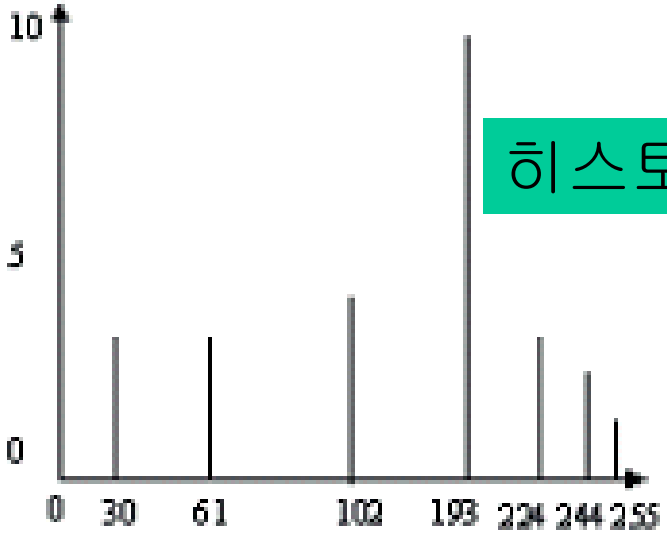
히스토그램 평활화 예

명암값	빈도수 Hist[j]	누적합 sum[i]	정규화 n[i]
40	3	3	30
50	3	6	61
60	4	10	102
70	9	19	193
80	3	22	224
90	2	24	244
100	1	25	255

정규화

30	30	61	102	193
30	61	102	193	193
61	102	193	193	224
102	193	193	224	244
193	193	224	244	255

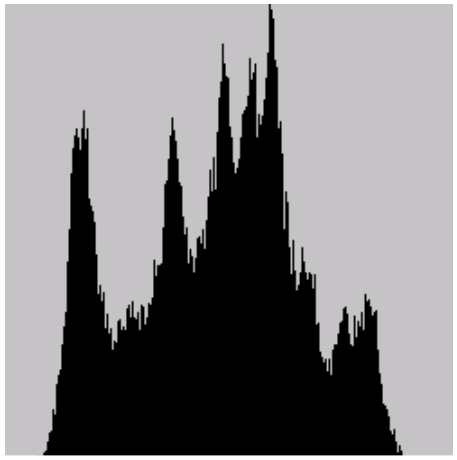
평활화된 영상



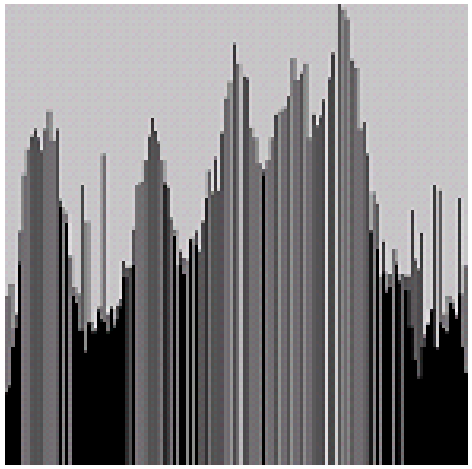
히스토그램

히스토그램 평활화 적용 예

원영상

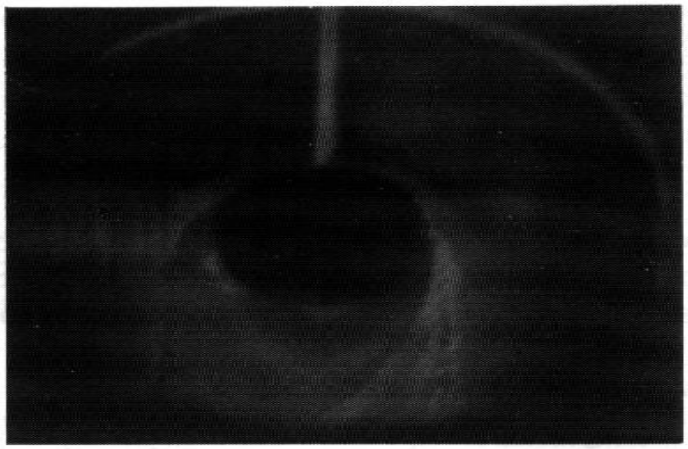


결과 영상

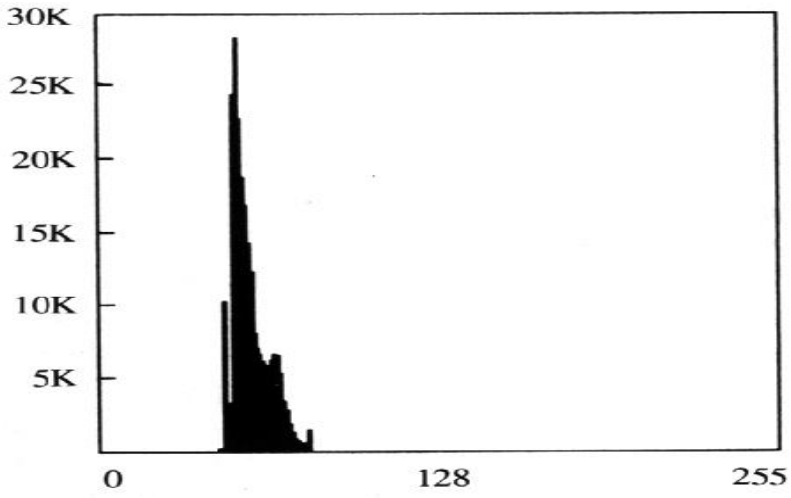


히스토그램 평활화

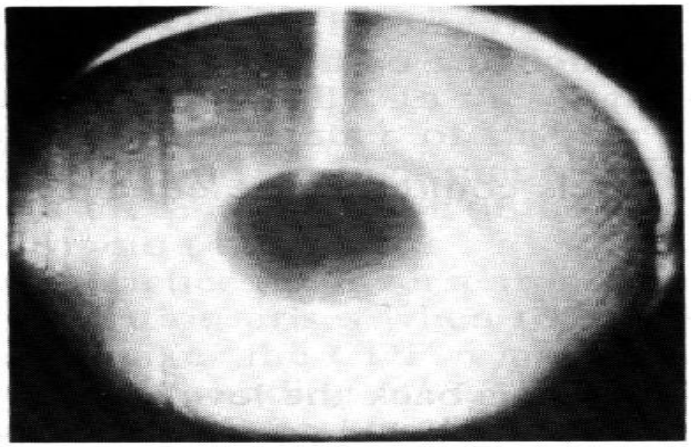
- 영상의 명암값이 <u>좁은 영역에 모여 있을</u> 때에 효과적

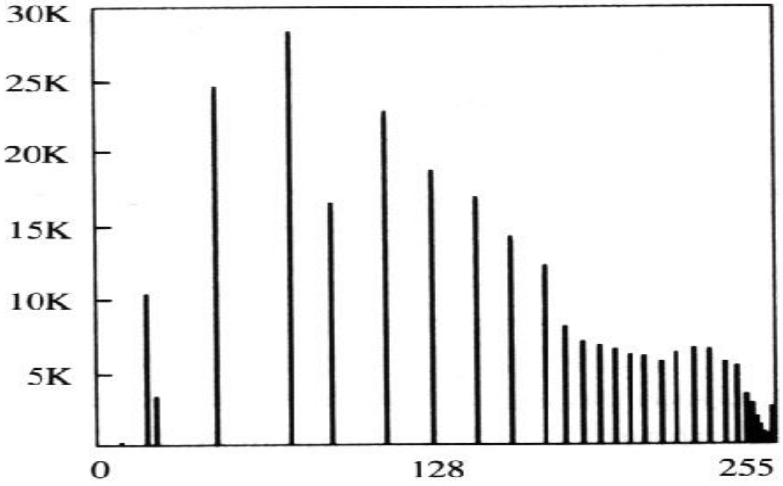


(a)

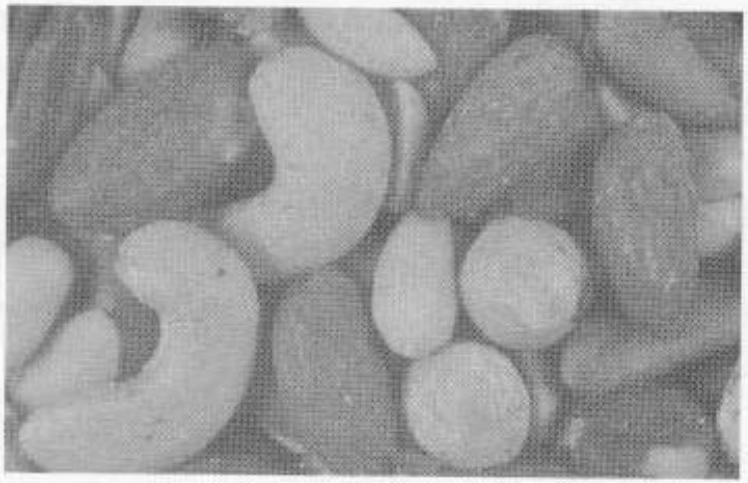


(b)

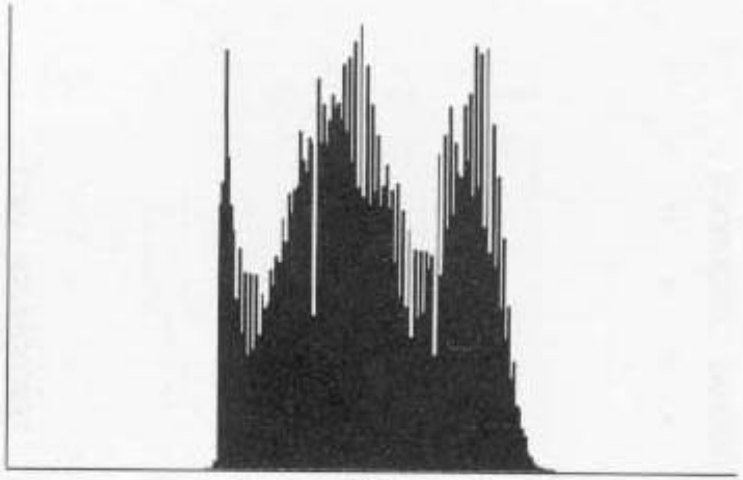




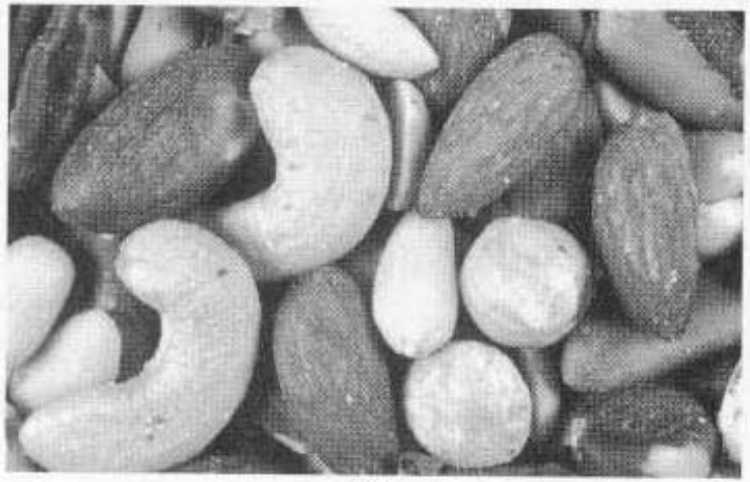
히스토그램 평활화



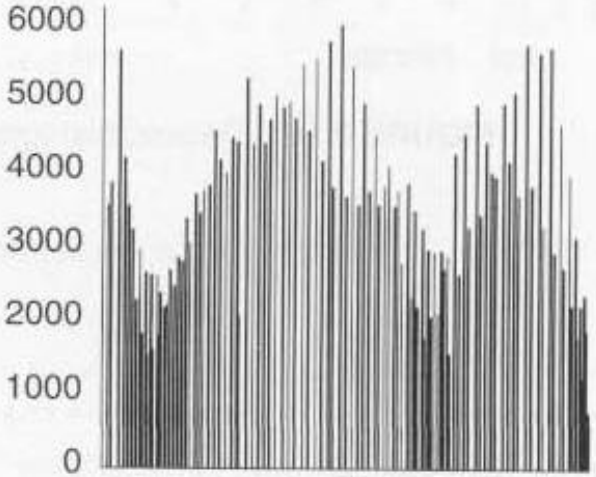
(a)



(b)



(c)



(d)

FIGURE 2.8 (a) Original image; (b) histogram of original image; (c) histogram equalized image; (d) histogram of equalized image.

명암 대비

- 낮은 명암 대비
  - 히스토그램이 일부분에 집중
- 높은 명암 대비
  - 히스토그램이 두개의 큰 마루를 가짐
- 좋은 명암 대비
  - 균일한 화소값 분포를 가짐
  - 특정한 마루나 골이 부각되지 않음

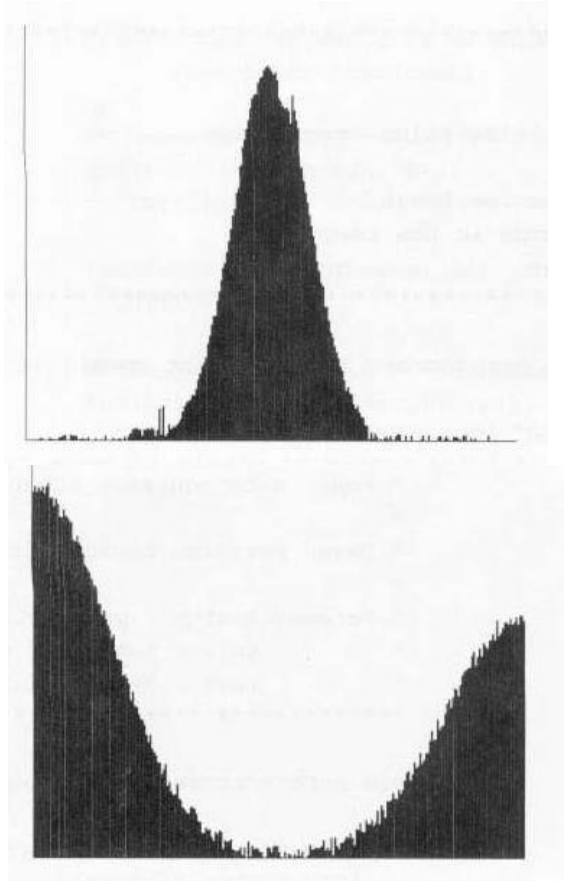


FIGURE 2.12 Low and high contrast histograms.

명암대비 스트레칭

- 히스토그램이 모든 범위의 화소 값을 포함하도록 영상을 확장

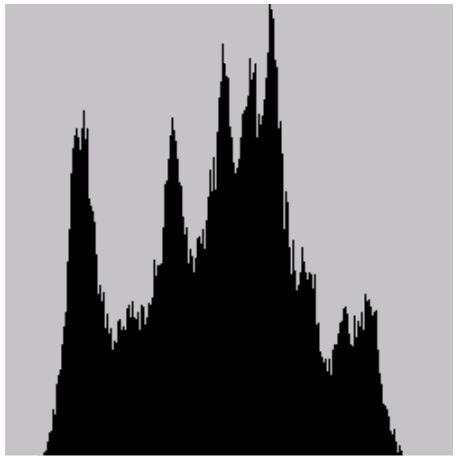
- 중앙에 집중된 히스토그램을 갖는 영상에 적합

$$P'(x,y) = \frac{P(x,y) - \min}{\max - \min} \times 255$$

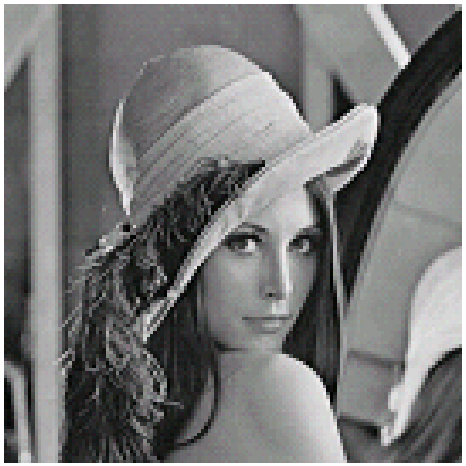
min : 최저 화소값, max : 최고 화소값

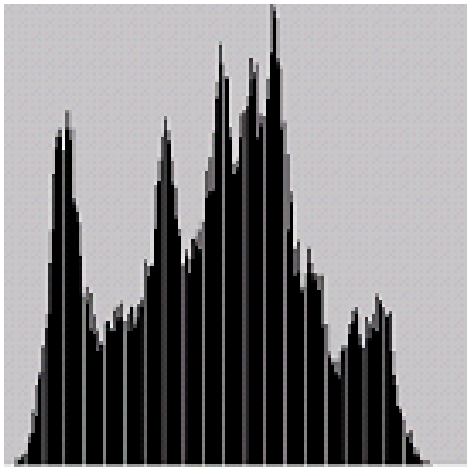
명암대비 스트레칭 예

원영상



결과 영상





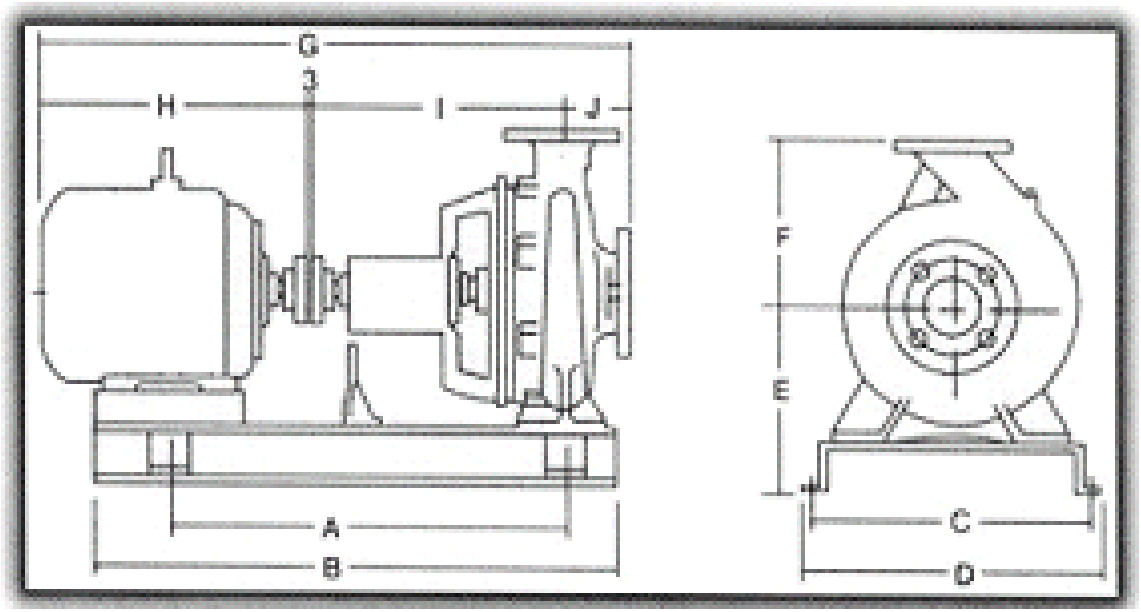
이진화

$$P'(x,y)=\begin{cases}255 & P(x,y) \geq T \\ 0 & P(x,y) < T\end{cases}$$

- 이진화
  - 2가지 값을 갖는 영상으로 변환함으로써 영상의 분석을 용이하게 할 수 있음

T = 128을 이용한 이진화 예

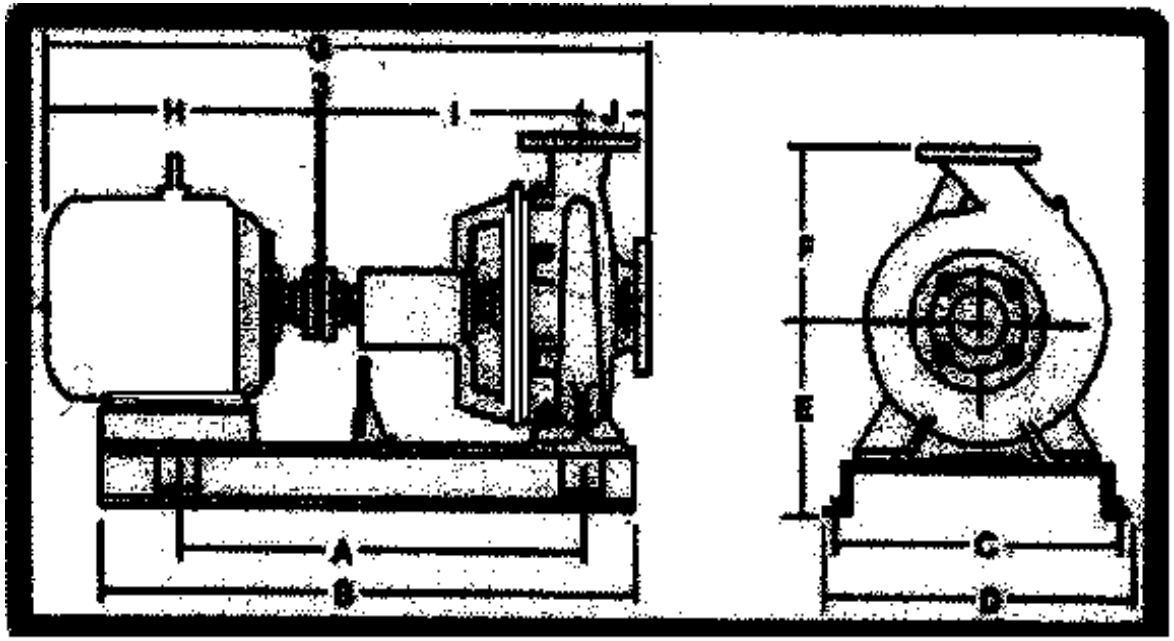
- **잡음 제거에 이용**



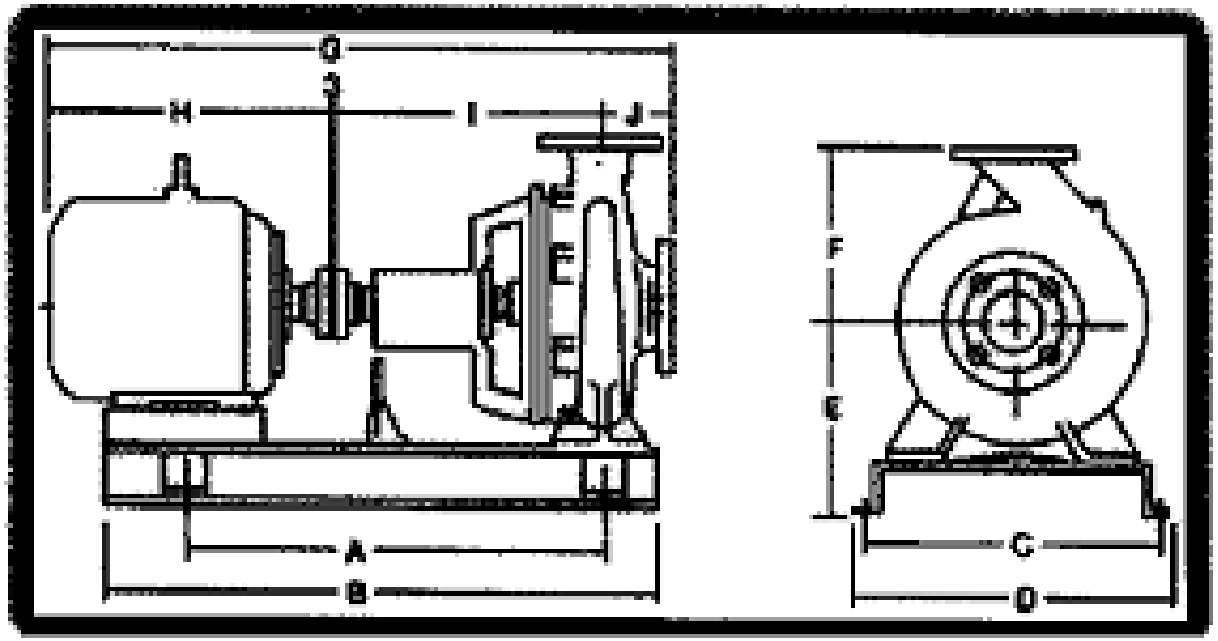
스캔된 영상

이진화

- T=255를 이용하여 이진화한 예



- T=230을 이용한 잡음 제거 예

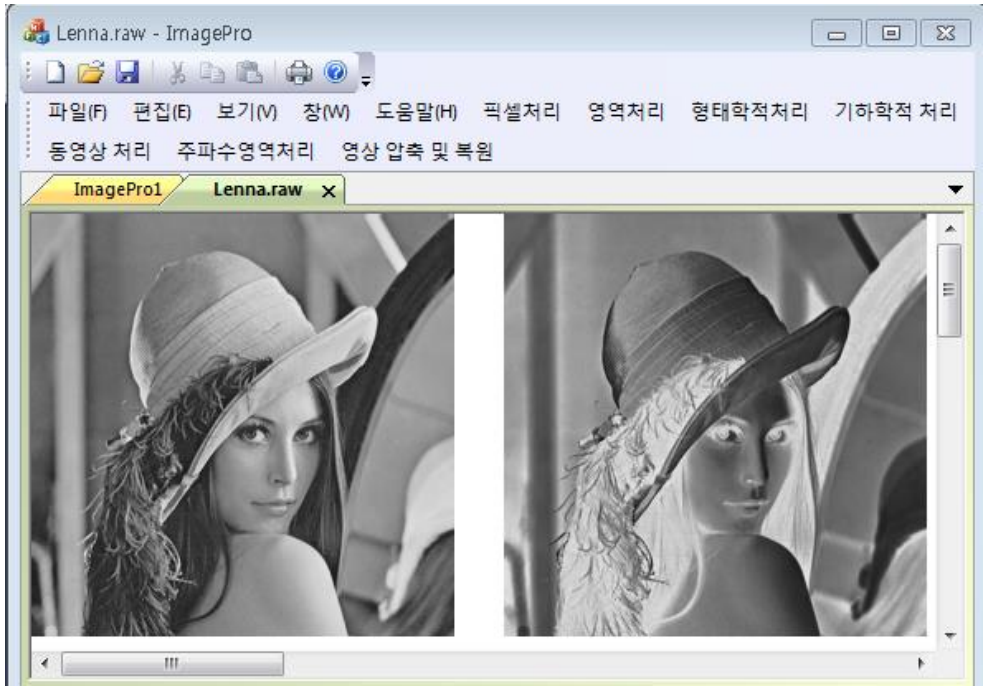


잡음이 제거된 영상

역상

- 픽셀의 값을 다음과 같이 변환하면 역상 영상을 얻을 수 있음

$$P'(x,y) = 255 - P(x,y)$$



영상 사이의 픽셀 기반 처리

- 2개 또는 그 이상의 서로 다른 영상을 포함하는 연산을 기반으로 하여 화소 값을 생성

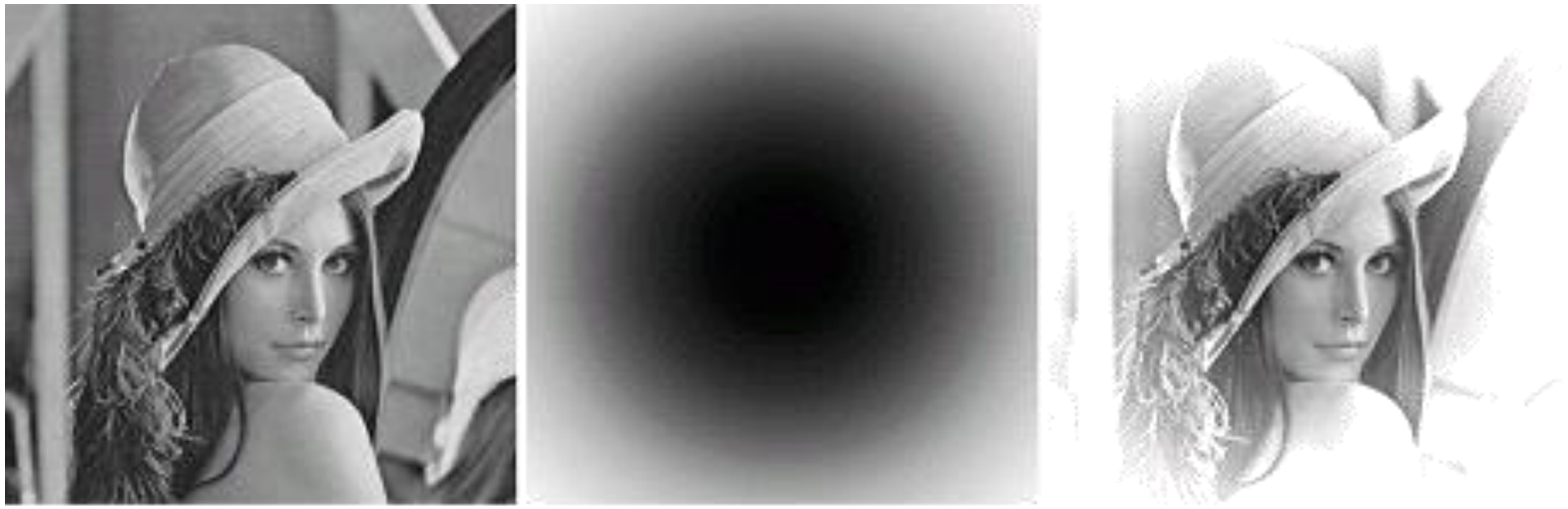
- 덧셈연산, 뺄셈연산, AND/OR 연산, 평균연산 등이 있음

덧셈 연산

- 두개의 영상에 덧셈연산을 수행하여 새로운 영상을 생성할 때 사용

- 일반적으로 다음과 같은 **혼합 함수** 사용

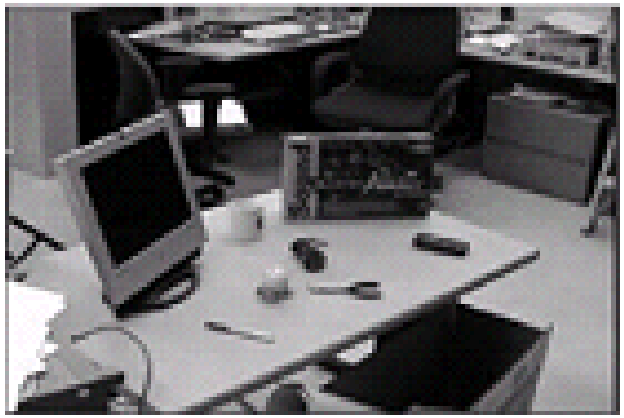
$$O(x,y) = \alpha \times I_1(x,y) + \beta \times I_2(x,y) \qquad \alpha = 1,\ \beta = 1$$



뺄셈 연산

- **보안** 시스템이나 영상 분석 시스템에서 이용

영상 1



영상 2

val = abs(영상 1 – 영상 2)

if (val > 64) val = 255
else val = 0

뺄셈 연산

영상 3

영상 4

침입자 감지

val = abs(영상 3 – 영상 4)

if (val > 64) val = 255
else val = 0

산술 덧셈 연산 구현

1. 영상 출력 프로그램으로부터 시작

- 1장에서 구현한 프로그램 사용

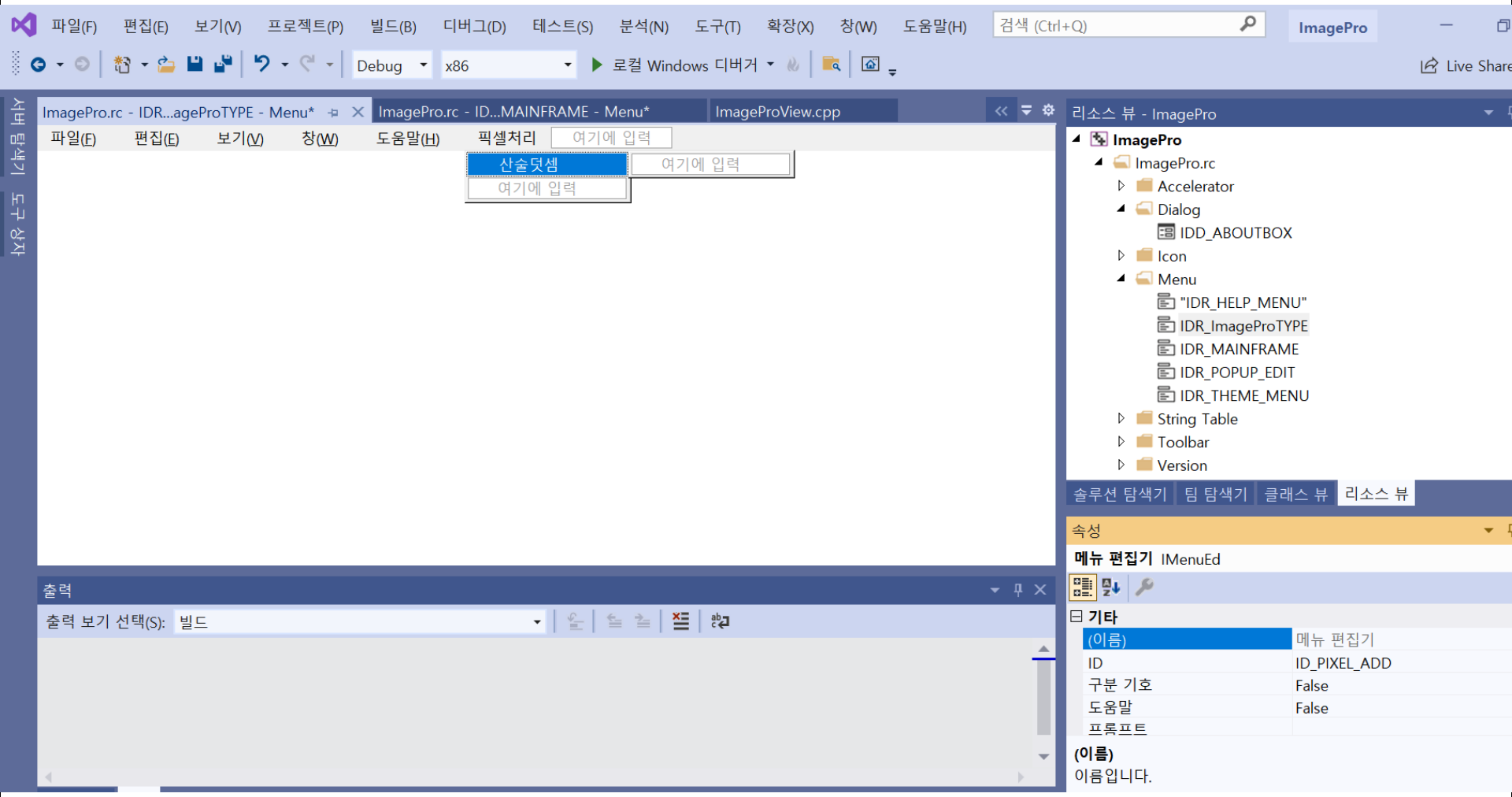
2. 메뉴 막대에 [픽셀처리] 메뉴 추가

- 리소스 편집기에서 메뉴 추가
- 이름 : 픽셀처리

3. 부메뉴 추가

- 이름 : 산술 덧셈
- ID : ID_PIXEL_ADD

산술 덧셈 연산 구현


파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 테스트(S) 분석(N) 도구(T) 확장(X) 창(W) 도움말(H)
검색 (Ctrl+Q)
ImagePro
Debug
x86
로컬 Windows 디버거
Live Share
서버 탐색기
도구 상자
ImagePro.rc - IDR...ageProTYPE - Menu*
ImagePro.rc - ID...MAINFRAME - Menu*
ImageProView.cpp
파일(F) 편집(E) 보기(V) 창(W) 도움말(H) 픽셀처리 여기에 입력
산술덧셈
여기에 입력
여기에 입력
출력
출력 보기 선택(S): 빌드
리소스 뷰 - ImagePro
ImagePro
ImagePro.rc
Accelerator
Dialog
IDD_ABOUTBOX
Icon
Menu
"IDR_HELP_MENU"
IDR_ImageProTYPE
IDR_MAINFRAME
IDR_POPUP_EDIT
IDR_THEME_MENU
String Table
Toolbar
Version
솔루션 탐색기 팀 탐색기 클래스 뷰 리소스 뷰
속성
메뉴 편집기 IMenuEd
기타
(이름) 메뉴 편집기
ID ID_PIXEL_ADD
구분 기호 False
도움말 False
프롬프트
(이름)
이름입니다.

4. 멤버 함수 추가

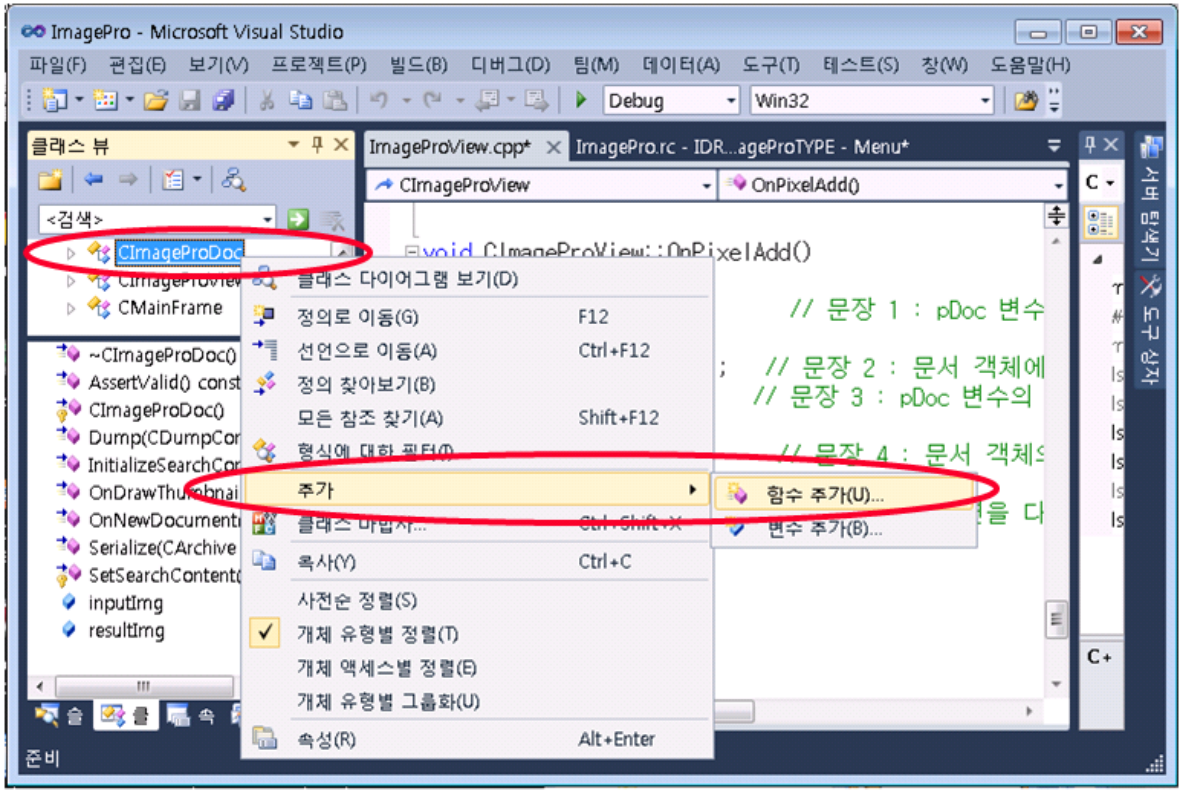
- 부메뉴 위에서 마우스 오른쪽 버튼으로 팝업메뉴가 나오게 함
- [이벤트 처리기 추가] 메뉴 선택
- [메시지형식] 상자에서 [COMMAND] 선택
- [클래스 목록] 상자에서 CImageProView 선택
- 함수 처리기 이름: OnPixelAdd
- [추가 및 편집] 버튼 선택

5. OnPixelAdd() 함수 편집

```
void CImageProView::OnPixelAdd()
{
        CImageProDoc* pDoc = GetDocument();
         ASSERT_VALID(pDoc);

         pDoc->PixelAdd();

         Invalidate(FALSE);
}
```

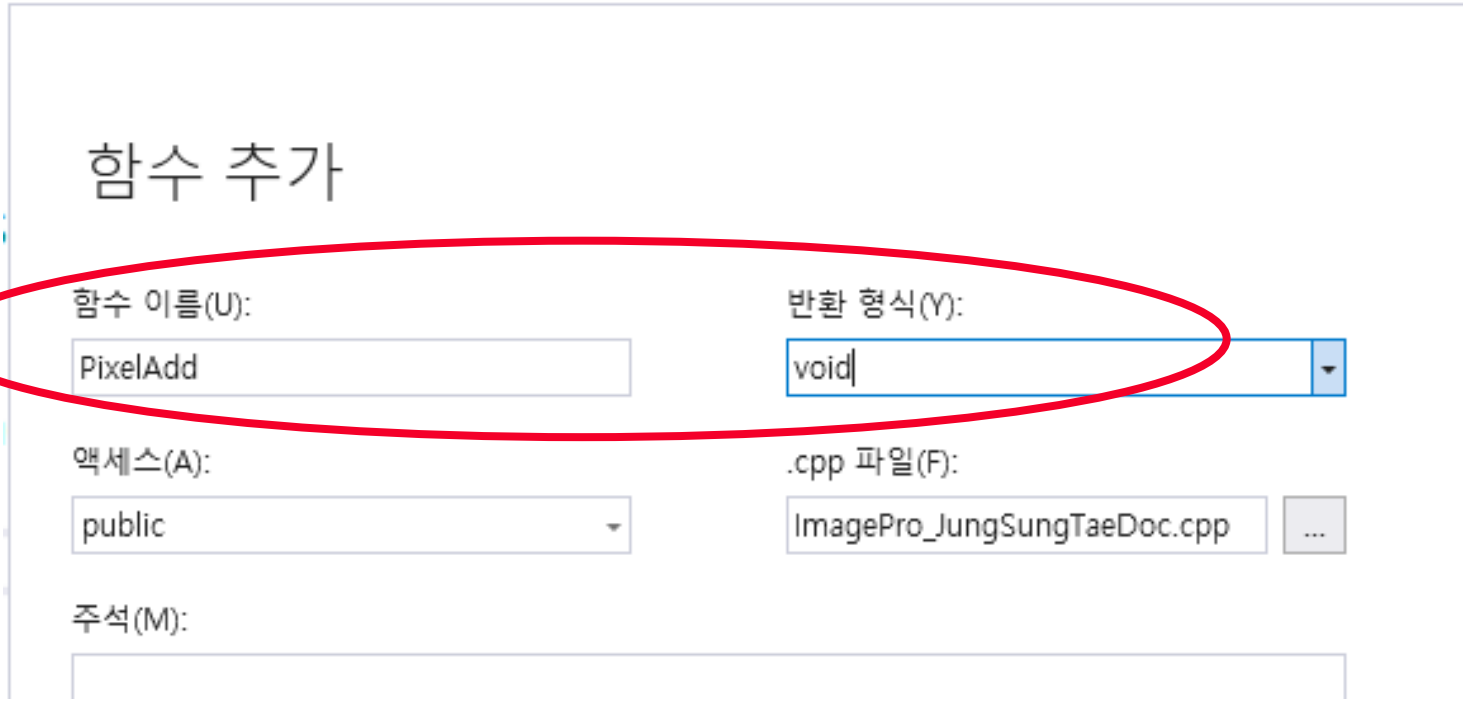
6. CImageProDoc 클래스에 PixelAdd() 함수 추가

- 클래스 뷰 에서 CImageProDoc 클래스를 마우스 오른쪽 버튼으로 선택하고 [추가] ➔ [함수추가] 선택



7. 반환형식과 함수이름 지정

- 함수 이름 : PixelAdd
- 반환 형식 : void



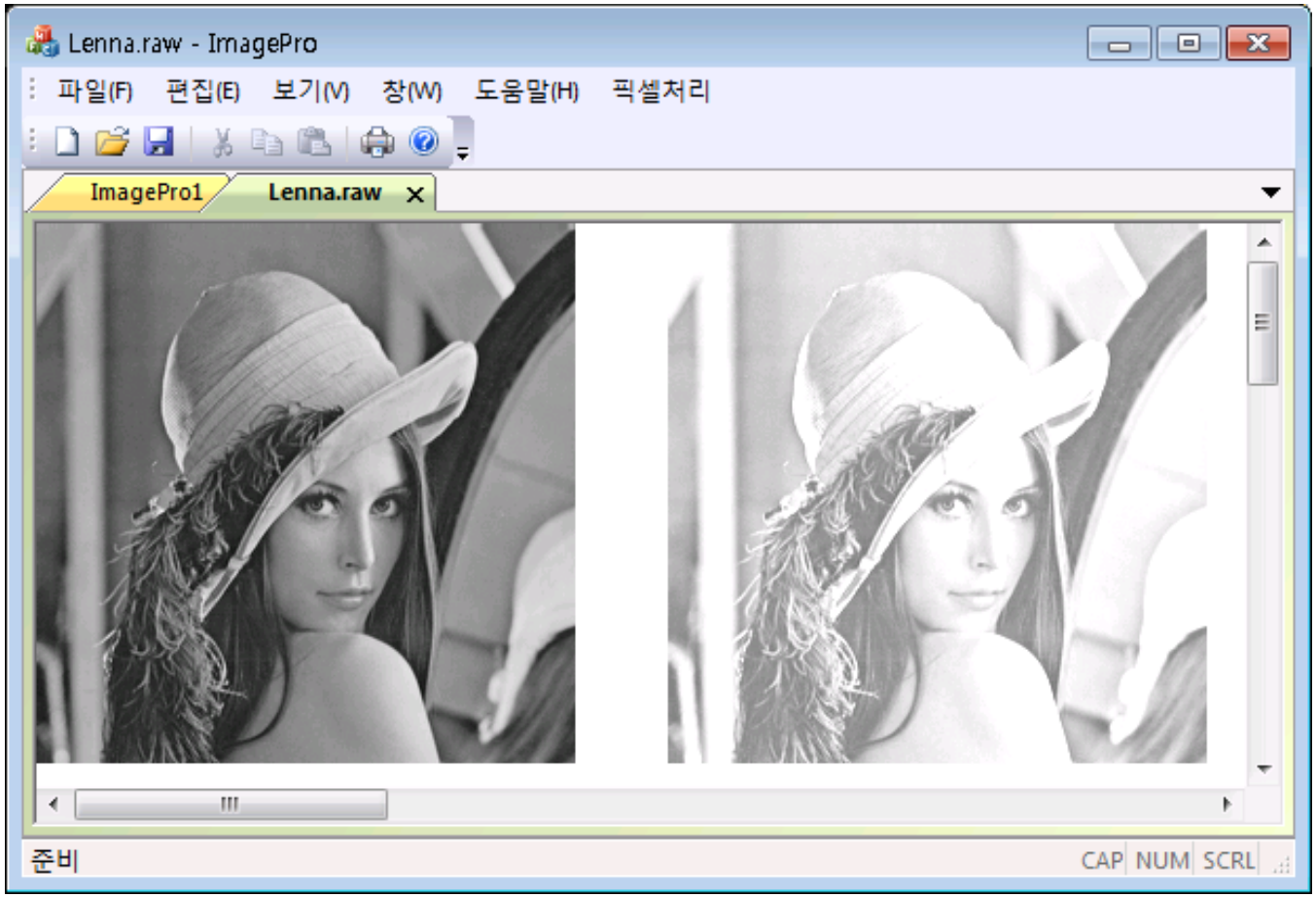
8. PixelAdd 함수 편집

```
void CImageProDoc::PixelAdd()
{
    int value=0;

    for(int x=0; x<256; x++)
        for(int y=0; y<256; y++) {
            value = inputImg[x][y]+100;          //원 이미지+100
            if(value > 255) resultImg[x][y]=255;
            else resultImg[x][y] = value;
        }
}
```

9. OnDraw() 함수 편집

```
void CImageProView::OnDraw(CDC* pDC)
{
   CImageProDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   if (!pDoc) return;

   for(int y=0; y<256; y++)                //Original Image 출력
      for(int x=0; x<256; x++) {
            pDC->SetPixel(x,y,RGB(pDoc->inputImg[y][x],
                        pDoc->inputImg[y][x], pDoc->inputImg[y][x]));
      }
   for(int y=0; y<256; y++)                //Result Image 출력
      for(int x=0; x<256; x++) {
         pDC->SetPixel(x+300,y,RGB(pDoc->resultImg[y][x],
                        pDoc->resultImg[y][x],  pDoc->resultImg[y][x]));
      }
}
```

10.프로그램을 컴파일하고 실행



히스토그램 평활화

- [픽셀 처리] 메뉴에 히스토그램 평활화를 위한 부메뉴 추가
  - 이름 : 히스토그램 평활화
  - ID : ID_PIXEL_HISTO_EQ

- 이벤트 처리기 마법사를 이용하여 OnPixelHistoEq() 함수 추가

```
void CImageProView::OnPixelHistoEq()
{
   CImageProDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);

   pDoc->PixelHistoEq();  // CImageProDoc 클래스의 PixelHistoEq() 호출
   Invalidate(FALSE);     //화면 갱신.
}
```

- CImageProDoc 클래스에 PixelHistoEq() 함수 추가

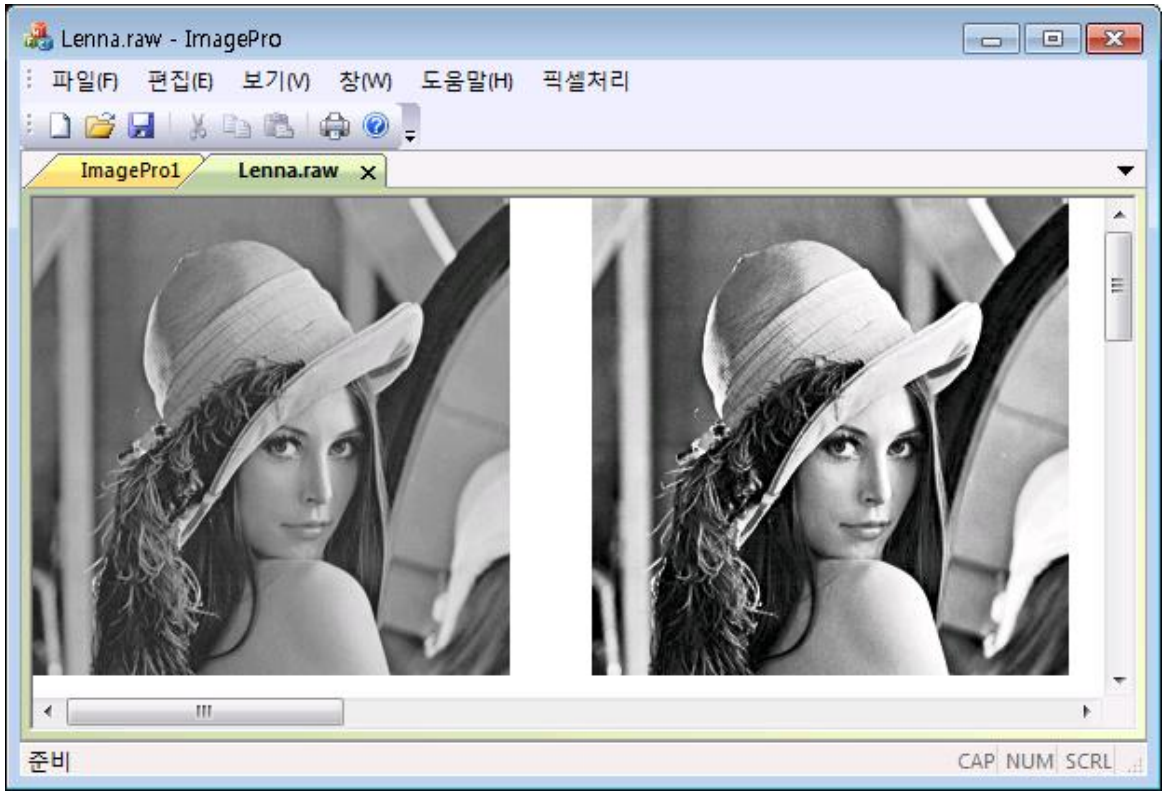
```
void CImageProDoc::PixelHistoEq()
{
    int x, y, i, k;
    int acc_hist = 0;      // 히스토그램의 합을 누적하는 변수
    float N = 256 * 256;   // 영상의 전체 픽셀 수
    int hist[256], sum[256];

    for(k=0; k<256; k++) hist[k]=0;

  // 명암값의 빈도수 조사
     for(y = 0; y < 256; y++)
        for(x = 0; x < 256; x++) {
           k = inputImg[y][x];
           hist[k] = hist[k]+1;
        }
```

히스토그램 평활화

```
// 누적된 히스토그램 합 계산
for(i=0; i<256; i++) {
   acc_hist = acc_hist + hist[i];
   sum[i] = acc_hist;
}

for(y = 0; y < 256; y++)
      for(x = 0; x < 256; x++) {
         k = inputImg[y][x];
         resultImg[y][x] = sum[k] / N * 255;
      }
}
```

히스토그램 평활화

- 프로그램을 컴파일하고 실행



두 영상의 산술 덧셈

- 두 개의 영상을 읽어 들일 기억장소가 필요
- CImageProDoc 클래스의 에 inputImg2 변수를 추가

```
class CImageProDoc : public CDocument
{
   ...
// Attributes
public:
   unsigned char inputImg[256][256];
   unsigned char inputImg2[256][256];
   unsigned char resultImg[256][256];
   ...
}
```

두 영상의 산술 덧셈

- [픽셀 처리] 메뉴에 두 영상의 산술 덧셈을 위한 메뉴 추가
  - 이름 : 두 영상의 산술 덧셈
  - ID : ID_PIXEL_TWO_IMAGE_ADD

- 클래스 마법사를 이용하여 OnPixelTwoImageAdd() 함수추가

```
void CImageProView::OnPixelTwoImageAdd()
{
  CImageProDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  pDoc->PixelTwoImageAdd();

  Invalidate(FALSE);
}
```

- CImageProDoc 클래스에 PixelTwoImageAdd() 함수 추가

```
void CImageProDoc::PixelTwoImageAdd()
{
   int value = 0;
   LoadTwoImages();
   for(int y=0; y<256; y++)
      for(int x=0; x<256; x++) {
         value = inputImg[y][x] + inputImg2[y][x];
         if (value > 255) resultImg[y][x] = 255;
         else resultImg[y][x] = value;
      }
}
```

- CImageProDoc 클래스에 LoadTwoImages() 함수 추가

```
void CImageProDoc::LoadTwoImages()
{
  CFile file;     // CFile 객체 선언
  CFileDialog dlg(TRUE);  // 파일 선택 대화상자 객체 선언
                          //       TRUE : 파일 열기
                          //       FLASE : 파일 저장

  AfxMessageBox("Select the First Image");

  if(dlg.DoModal()==IDOK)   {  // 파일 선택 대화 상자 실행
    file.Open(dlg.GetPathName(), CFile::modeRead);   // 파일 열기
    file.Read(inputImg, 256*256);       // 영상 읽기
    file.Close();             // 파일 닫기
  }
```

두 영상의 산술 덧셈

```
AfxMessageBox("Select the Second Image");

  if(dlg.DoModal()==IDOK)   {   // 파일 선택 대화 상자 실행
    file.Open(dlg.GetPathName(), CFile::modeRead);   // 파일 열기
    file.Read(inputImg2, 256*256);        // 영상 읽기
    file.Close();              // 파일 닫기
  }
}
```

두 영상의 산술 덧셈

- 두 영상을 이용한 처리의 경우 영상을 세 개의 영상을 출력해야 함
  - 한 영상을 사용하는 경우와 두 영상을 사용하는 경우를 구분할 필요가 있음
    - 변수를 이용하여 구분
      - int viewMode;
    - viewMode 변수의 값을 기호 상수로 정의하여 사용
      - TWO_IMAGES : 출력할 영상이 2개
      - THREE_IMAGES : 출력할 영상이 3개

- CImageProView 클래스에 출력 모드를 구분하기 위한 변수 추가

```
class CImageProView : public CScrollView
{
   …
// Attributes
public:
   CImageProDoc* GetDocument();
   int viewMode;          // 영상 출력 모드
   ...
}
```

두 영상의 산술 덧셈

- viewMode 변수 값을 THREE_IMAGES로 설정
  - OnPixelTwoImagesAdd() 함수를 수정

```
void CImageProView::OnPixelTwoImageAdd()
{
   CImageProDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   pDoc->PixelTwoImageAdd();

   viewMode = THREE_IMAGES;

   Invalidate(FALSE);
}
```

두 영상의 산술 덧셈

- 기존의 입력 영상을 한 개만 사용하는 함수
  - "viewMode = TWO_IMAGES;"라는 문장을 추가

```
void CImageProView::OnPixelAdd()
{
   CImageProDoc* pDoc;    // 문장 1 : pDoc 변수 선언

   pDoc = GetDocument();   // 문장 2 : 문서 객체에 대한 포인터 획득
   ASSERT_VALID(pDoc);    // 문장 3 : pDoc 변수의 오류 검증
   pDoc->PixelAdd();     // 문장 4 : 문서 객체의 PointAdd() 함수 호출

   viewMode = TWO_IMAGES;
   Invalidate(FALSE);     // 문장 5 : 화면을 다시 그림
}
```

두 영상의 산술 덧셈

```
void CImageProView::OnPixelHistoEq()
{
    CImageProDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    pDoc->PixelHistoEq();   // CImageProDoc 클래스의
                            // PixelHistoEq()함수 호출

    viewMode = TWO_IMAGES;
    Invalidate(FALSE);   // 화면 갱신
}
```

두 영상의 산술 덧셈

- viewMode 변수의 값에 따라 입력 영상과 결과 영상을 다르게 출력하도록 OnDraw() 함수를 편집

```
void CImageProView::OnDraw(CDC* pDC)
{
   CImageProDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);

   for(int y=0; y<256; y++)      // 입력 영상 출력
       for(int x=0; x<256; x++)
           pDC->SetPixel(x, y, RGB(pDoc->inputImg[y][x],
                                   pDoc->inputImg[y][x],
                                   pDoc->inputImg[y][x]));
```

두 영상의 산술 덧셈

```
if (viewMode == THREE_IMAGES) {
   for(int y=0; y<256; y++)      // 두번째 입력 영상 출력
     for(int x=0; x<256; x++)
       pDC->SetPixel(x+300, y, RGB(pDoc->inputImg2[y][x],
                                   pDoc->inputImg2[y][x],
                                   pDoc->inputImg2[y][x]));

   for(int y=0; y<256; y++)      // 결과 영상 출력
     for(int x=0; x<256; x++)
       pDC->SetPixel(x+600, y, RGB(pDoc->resultImg[y][x],
                                   pDoc->resultImg[y][x],
                                   pDoc->resultImg[y][x]));
}
```

두 영상의 산술 덧셈

```
    else {
        for(int y=0; y<256; y++)      // 결과 영상 출력
          for(int x=0; x<256; x++)
            pDC->SetPixel(x+300, y, RGB(pDoc->resultImg[y][x],
                                        pDoc->resultImg[y][x],
                                        pDoc->resultImg[y][x]));
    }
}
```

두 영상의 산술 덧셈

- CImageProView.cpp 파일 상단에 상수 TWO_IMAGES와 THREE_IMAGES를 정의

```
// ImageProView.cpp : implementation of the CImageProView class
//

#include "stdafx.h"
#include "ImagePro.h"

…

#define TWO_IMAGES      1
#define THREE_IMAGES    2
```

두 영상의 산술 덧셈

- 프로그램을 컴파일하고 실행
  - Lenna.raw 영상과 two_images_add_mask.raw 영상 이용