제 3 장 영역 기반 처리

학습목표

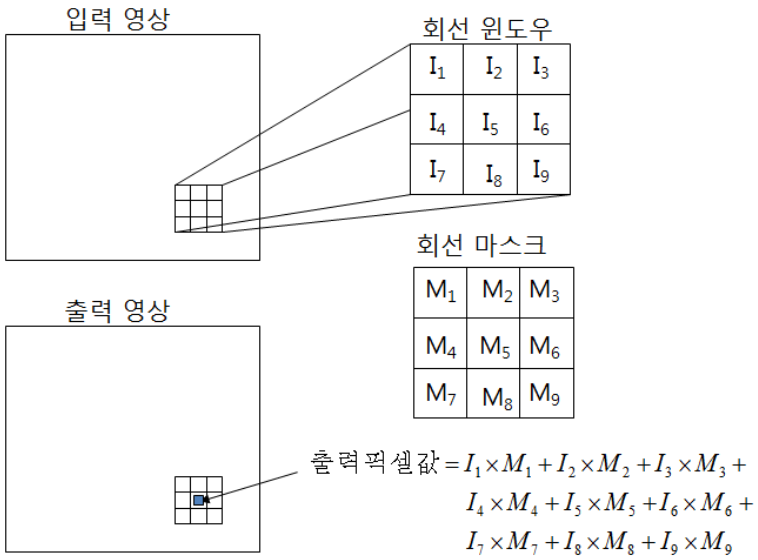
- 회선의 개념을 설명할 수 있다.
- 다음 영역기반 처리의 원리를 설명하고 프로그램을 작성할 수 있다
  - 영상 흐리게 하기
  - 선명화
  - 경계선 검출
  - 잡음제거
  - 컬러 영상 처리
- 이중포인터를 설명하고 프로그램에서 활용할 수 있다
- 프로그램에서 PGM, PPM 파일을 활용할 수 있다

영역 기반 처리

- 입력 화소와 그 주위 화소를 이용하여 출력 화소값을 결정
- 회선(convolution) 기법을 널리 이용
- 영역 기반 처리 예
  - 흐리게 하기, 선명하게 하기, 경계선 검출, 잡음 제거

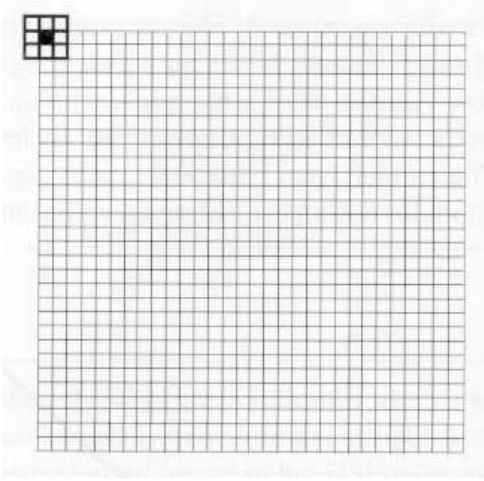
회선

- 출력 픽셀 값
  - 입력 픽셀과 그 주위 픽셀 값에 회선 마스크의 값을 곱하여 합한 값



회선 수행 방법

- 좌측 상단의 픽셀부터 한 픽셀 씩 차례로 수행
  - 먼저 우측 방향으로 진행
  - 한 줄이 끝나면 아래 줄로 이동

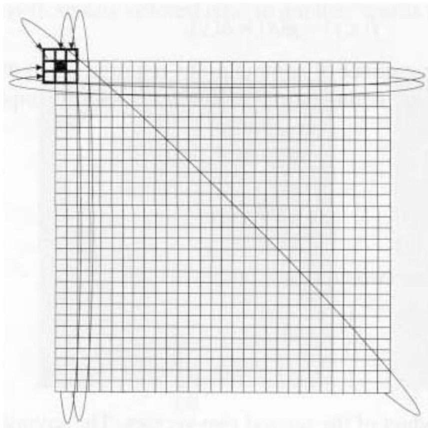


회선 마스크의 특성

- 회선 마스크의 크기는 홀수를 사용
  - 주위 픽셀 값을 각 방향에 대칭적으로 고려해야 하므로
  - 3 x 3, 5 x 5, 7 x 7 등의 크기 사용
- 많은 회선 마스크들은 계수들의 합이 1 이됨
  - 회선된 영상은 원영상과 같은 평균 밝기 값을 가짐
- 경계선 검출등 일부 회선 마스크에서는 음수의 계수를 포함하고 계수 합이 0 이됨
  - 음의 화소값들이 생성될 수 있으므로 전형적으로 생성된 화소값에 일정한 상수(최대밝기/2 와 같은)가 더해짐

영상의 경계처리

- 0 삽입
  - 윈도우의 빈 셀들의 계수를 <u>0으로</u> 가정함
- 윈도우가 영상과 중첩되는 첫 위치에서 회선이 시작
  - 마스크가 3X3일 때에 (0,0) 대신 <u>(1,1)에서</u> 시작
- 원영상의 크기를 조정
  - 영상의 경계부분의 화소들을 <u>복사</u>
  - 영상을 둘러쌈



영상 흐리게하기

- 입력 픽셀 값을 주위 픽셀 값들과의 <u>평균 값</u>으로 변환하는 다음과 같은 회선 마스크를 널리 사용

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$

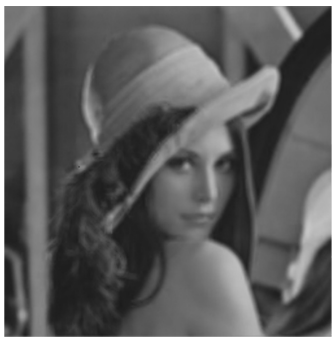
평균 마스크

영상 흐리게 하기 예

입력영상

3X3 마스크 적용

5X5 마스크 적용



영상의 선명화

- 선명한 영상 생성을 위한 회선 마스크

0	−1	0
−1	5	−1
0	−1	0

마스크 1

−1	−1	−1
−1	9	−1
−1	−1	−1

마스크 2

선명화 적용 예

입력 영상

마스크 1 적용

마스크 2 적용

선명화 적용 예

10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10
10	10	10	10	20	10	10	10	10
10	10	10	20	30	20	10	10	10
10	10	20	30	40	30	20	10	10
10	10	10	20	30	20	10	10	10
10	10	10	10	20	10	10	10	10
10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10

0	10	10	10	10	10	10	10	10
10	10	10	10	0	10	10	10	10
10	10	10	10	40	10	10	10	10
10	10	0	20	50	20	0	10	10
10	0	40	50	80	50	40	0	10
10	10	0	20	50	20	0	10	10
10	10	10	10	40	10	10	10	10
10	10	10	10	0	10	10	10	10
10	10	10	10	10	10	10	10	10

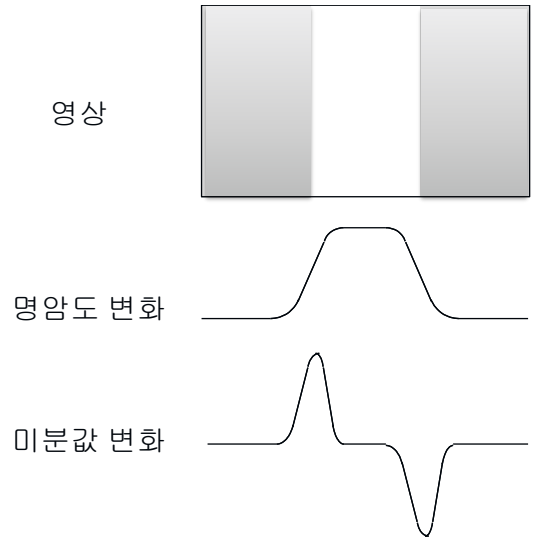
선명화 적용 예

경계선 검출

경계선

- 입력 영상에 대한 많은 정보 포함
- 물체를 식별하고 물체의 위치, 모양, 크기 등을 인지하는 데 큰 역할
- 영상의 밝기가 낮은 값에서 높은 값으로 또는 높은 값에서 낮은 값으로 변하는 지점에 존재

미분 연산자



경계선 검출 회선 마스크

- 미분 연산을 회선 마스크로 표현 가능
  - 수평 경계선과 수직 경계선을 개별적으로 검출

	수평 경계선	수직 경계선
Prewitt	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$
Roberts	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Sobel	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$

경계선 크기 계산

- 픽셀 ***I(x,y)***가 경계선일 가능성의 크기

$$E(x,y)=\sqrt{E_r^2(x,y)+E_c^2(x,y)}$$

$E_r(x,y)$: 수평 경계선 검출용 회선 마스크 적용 결과 값

$E_c(x,y)$: 수직 경계선 검출용 회선 마스크 적용 결과 값

경계선 검출 결과

입력영상

Sobel

Prewitt

Roberts

잡음 제거

잡음

- 가우시안 잡음
  - 정규 분포를 갖는 잡음
  - 영상의 픽셀 값으로부터 불규칙적으로 벗어나지만 뚜렷하게 벗어나지 않는 잡음

- 임펄스 잡음
  - 영상의 픽셀 값과는 뚜렷하게 다른 픽셀 값에 의한 잡음
    - 0, 255와 같은 뚜렷하게 잘못된 밝기 값을 갖는 화소

잡음 예

가우시안 잡음

임펄스 잡음

잡음 제거

평균 마스크

- 가우시안 노이즈를 줄이는데 효과적
- 임펄스 노이즈에는 비효과적
- 영상의 대비를 약화시킴

잡음 제거 적용 예

평균 마스크를 이용한 잡음 제거 결과

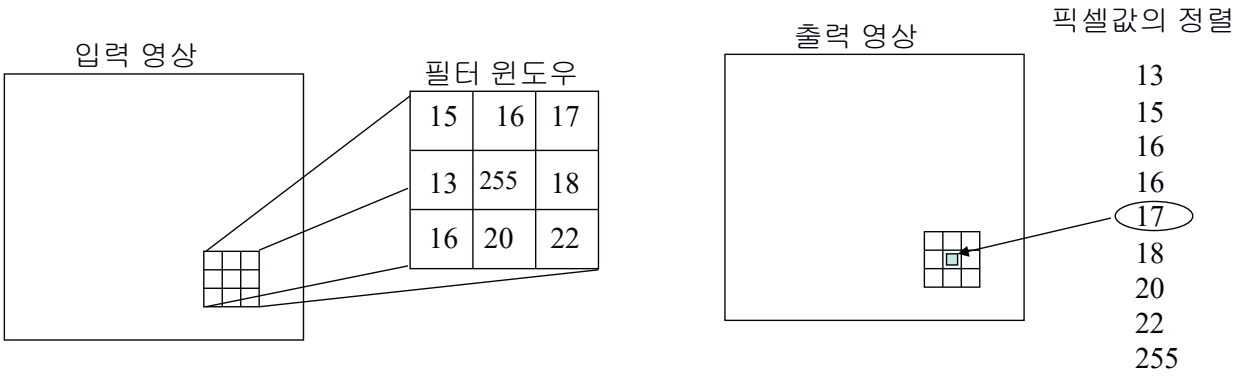
가우시안 잡음 제거 결과

임펄스 잡음 제거 결과

잡음 제거

중간값 필터링

- 임펄스 잡음을 제거하기위한 효과적인 방법
- 경계선을 보존 또는 강화



잡음 제거 적용 예

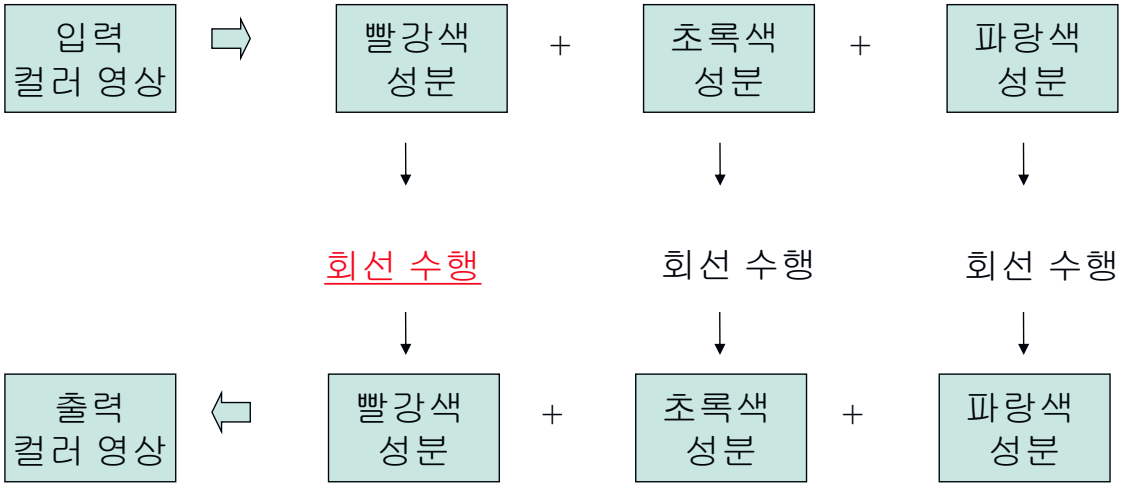
중간값 필터링을 이용한 잡음 제거 결과

가우시안 잡음 제거 결과

임펄스 잡음 제거 결과

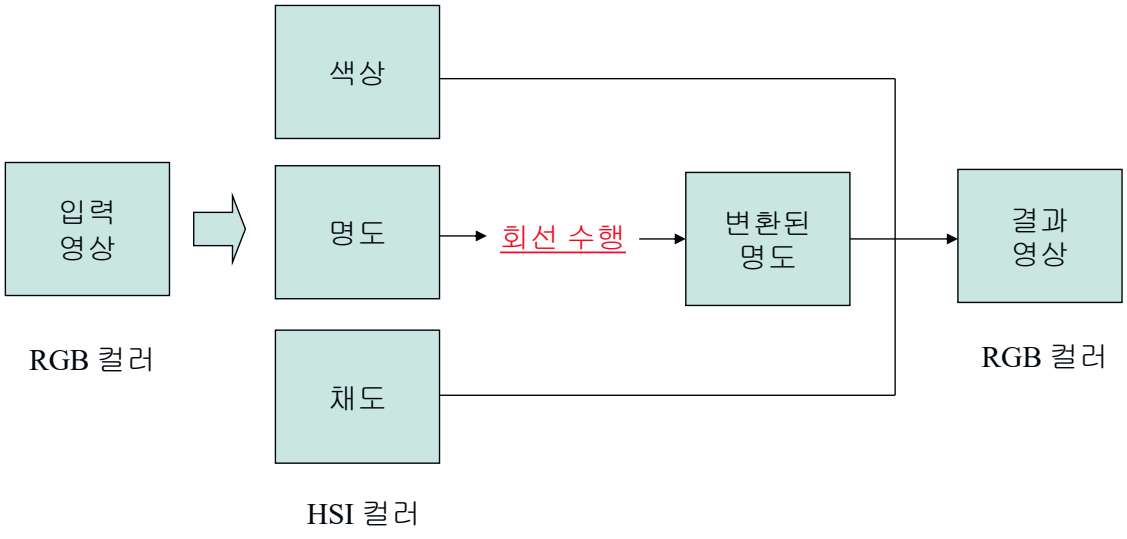
컬러 영상에서의 회선

RGB 컬러 모델에서의 회선



컬러 영상에서의 회선

HSI 컬러 모델에서의 회선



컬러 영상에서의 경계선 검출

- RGB 컬러 모델 사용
  - R,G,B 각각에 대하여 회선 수행

$$E(x,y)=\sqrt{E_{red}^2(x,y)+E_{green}^2(x,y)+E_{blue}^2(x,y)}$$

- HSI 컬러 모델
  - RGB 모델을 HSI 모델로 변환한 다음에 명도값 (I) 에 대해서만 회선 적용

임의의 크기 영상 처리를 위한 기억 장소 할당

- 프로그램의 유용성 향상을 위해 임의의 크기 처리 필요
- 임의의 크기 영상을 위해 PGM과 PPM 파일 사용

영상 저장을 위한 변수 선언

- **임의의 영상을 저장하기 위한 기억 장소**
  - 크기가 미리 정해져 있지 않으므로 배열 사용이 불가능
  - 기억 장소를 프로그램 실행 중에 동적으로 할당
  - 변수는 포인터로 선언

영상 저장을 위한 포인터 선언 방법

- **단일 포인터를 이용한 방법**
  - 일차원 배열처럼 사용
  - 변수 선언 및 기억 장소 할당이 단순
  - 영상의 픽셀 값을 사용하는 데에는 불편

단일 포인터를 이용한 방법

- 영상을 저장할 변수 필요
- 영상의 크기를 저장할 변수 필요
- 흑백 영상과 컬러 영상을 구분하기 위한 변수 필요

```
unsigned char *inputImg;   // 입력 영상의 기억 장소에 대한 포인터 변수
unsigned char *resultImg;  // 출력 영상의 기억 장소에 대한 포인터 변수
int imageWidth;          // 영상의 가로 크기
int imageHeight;          // 영상의 세로 크기
int depth;               // 1 = 흑백 영상, 3 = 컬러 영상
```


단일 포인터를 이용한 방법

- 기억 장소 할당

```
inputImg = (unsigned char *) malloc(imageWidth * imageHeight * depth);
resultImg = (unsigned char *) malloc(imageWidth * imageHeight * depth);
```

입력 픽셀값	저장 장소
I(0,0)	**inputImg[0]**
I(0,1)	**inputImg[1]**
...	...
I(0,imageWidth-1)	**inputImg[imageWidth-1]**
I(1,0)	**inputImg[imageWidth]**
I(1,1)	**inputImg[imageWidth+1]**
...	...
I(y,x)	**inputImg[y * imageWidth + x]**
...	...
I(imageHeight-1, imageWidth-1)	**inputImg[imageHeight * imageWidth - 1]**

흑백 영상

단일 포인터를 이용한 방법

- 임의의 크기의 흑백 영상에 대한 산술 덧셈

```
for (y = 0; y < imageHeight; y++)
   for (x = 0; x < imageWidth; x++) {
      value = inputImg[y * imageWidth + x] + 100;
      if (value > 255) value = 255;
      resultImg[y * imageWidth + x] = value;
   }
```


단일 포인터를 이용한 방법

- 단일 루프를 이용한 방법(흑백영상)

```
for (k = 0; k < imageWidth * imageHeight; k++) {
   value = inputImg[k] + 100;
   if (value > 255) value = 255;
   resultImg[k] = value;
}
```

이중 포인터를 이용한 방법

- 이중 포인터를 이용한 방법
  - 이차원 배열처럼 사용
  - 변수 선언과 기억 장소 할당이 어려움
  - 영상의 픽셀 값 사용이 편리

이중 포인터를 이용한 방법

- 영상 저장을 위한 변수 선언

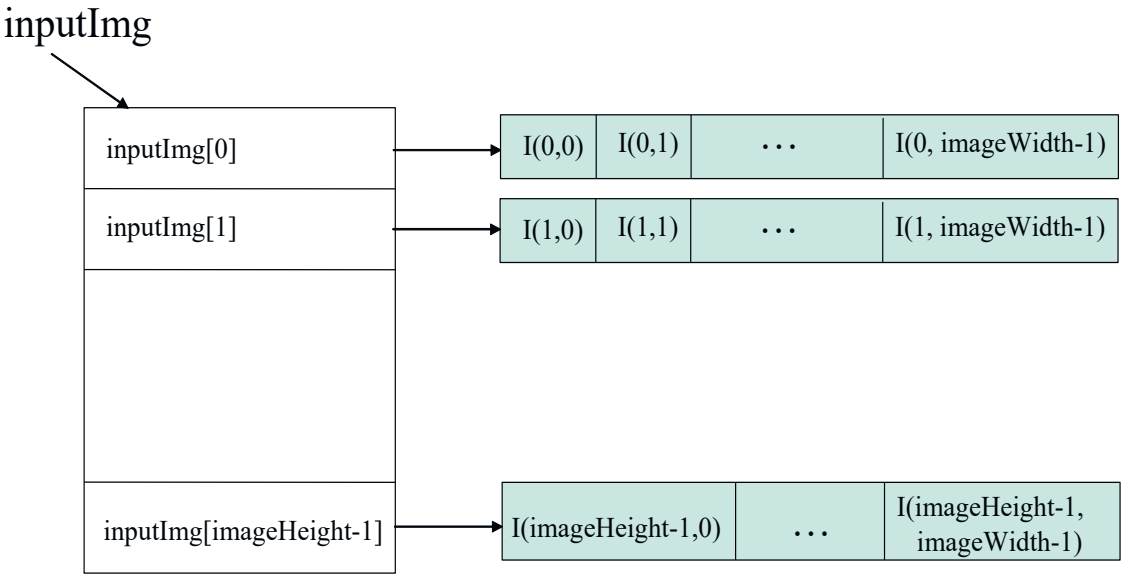
```
unsigned char **inputImg;  // 입력 영상의 기억 장소에 대한 포인터 변수
unsigned char **resultImg;  // 출력 영상의 기억 장소에 대한 포인터 변수
```

- 기억 장소 할당

```
inputImg = (unsigned char **) malloc(imageHeight * sizeof(unsigned char *));
resultImg = (unsigned char **) malloc(imageHeight * sizeof(unsigned char *));


for (i = 0; i < imageHeight; i++) {
  inputImg[i] = (unsigned char *) malloc(imageWidth * depth);
  resultImg[i] = (unsigned char *) malloc(imageWidth * depth);
}
```

이중 포인터를 이용한 방법



이중 포인터를 이용한 방법

- 흑백 영상의 산술 덧셈
  - 이차원 배열을 이용한 방법과 동일

```
for (y = 0; y < imageHeight; y++)
   for (x = 0; x < imageWidth; x++) {
      value = inputImg[y][x] + 100;
      if (value > 255) value = 255;
      resultImg[y][x] = value;
   }
```

PGM과 PPM 파일 형식 읽기 실습

PGM과 PPM 파일 형식 읽기

영상 저장을 위한 변수 선언 수정

-- 수정전

```
unsigned char  inputImg[256][256];
unsigned char  inputImg2[256][256];
unsigned char  resultImg[256][256];
```

-- 수정후

```
unsigned char **inputImg;  // 입력 영상의 기억 장소에 대한 포인터 변수
unsigned char **inputImg2;  // 입력 영상의 기억 장소에 대한 포인터 변수
unsigned char **resultImg;  // 출력 영상의 기억 장소에 대한 포인터 변수
int imageWidth;     // 영상의 가로 크기
int imageHeight;     // 영상의 세로 크기
int depth;        // 1 = 흑백 영상, 3 = 컬러 영상
```

PGM과 PPM 파일 형식 읽기

- 입력 영상 저장을 위한 포인터 변수 초기화
  - 기억 장소를 할당했는지의 여부 확인을 위해 사용

```
CImageProDoc::CImageProDoc(void)
{
  // TODO: add one-time construction code here
  inputImg = NULL;
  inputImg2 = NULL;
  resultImg = NULL;
}
```

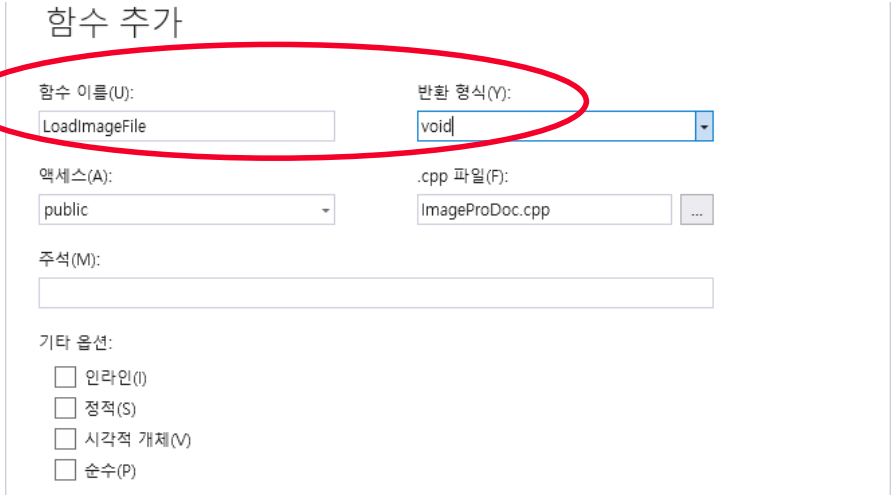

PGM과 PPM 파일 형식 읽기

- Serialize() 함수 수정

```
void CImageProDoc::Serialize(CArchive& ar)
{
  if (ar.IsStoring())
  {
  }
  else
  {
    LoadImageFile(ar);
  }
}
```

PGM과 PPM 파일 형식 읽기

- LoadImageFile() 함수를 CImageProDoc 클래스에 추가
  - 함수 이름은 LoadImageFile 반환 형식은 void



PGM과 PPM 파일 형식 읽기

- LoadImageFile() 함수는 매개변수를 필요로 함
  - 함수 추가 대화상자에 의해 함수 정의와 함수 선언이 프로그램에 추가됨
  - 추가된 함수 정의

```
void CImageProDoc::LoadImageFile()
{
// TODO: 여기에 구현 코드 추가.
}
```

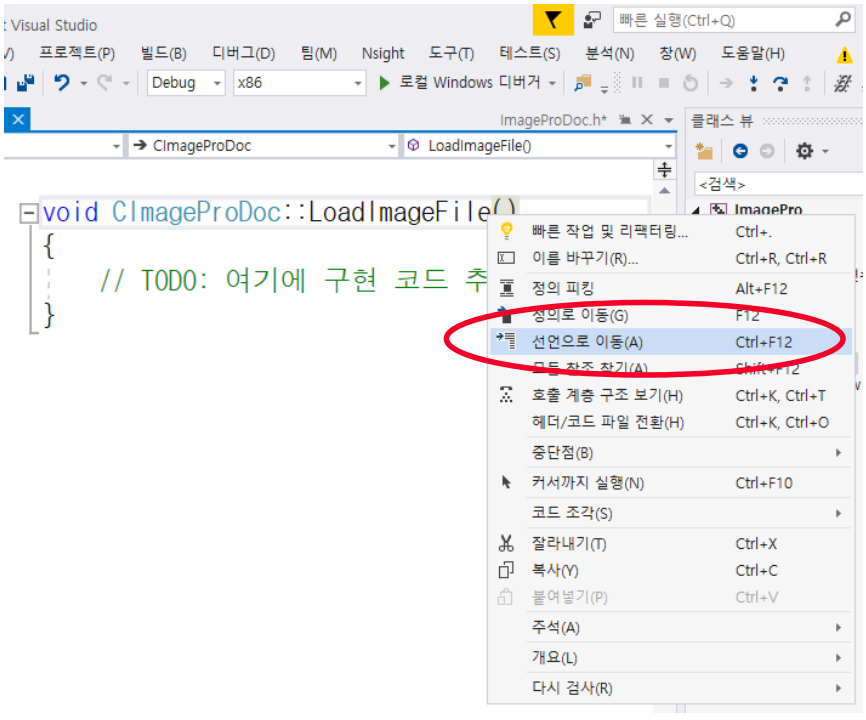
PGM과 PPM 파일 형식 읽기

- 추가된 함수 정의에 매개 변수 추가
  - 매개변수 형식 : CArchive&
  - 매개변수 이름 : ar

```
void CImageProDoc::LoadImageFile(CArchive &ar)
{
// TODO: 여기에 구현 코드 추가.
}
```


PGM과 PPM 파일 형식 읽기

- 추가된 함수 선언은 ImageProDoc.cpp 파일에 포함되어 있는 CImageProDoc 클래스 에 추가됨
- 함수이름 위에서 마우스 오른쪽 버튼을 클릭하여 [선언으로 이동]을 선택하면 함수 선언으로 이동 가능



PGM과 PPM 파일 형식 읽기

- 추가된 함수 선언으로 이동하면 다음과 같이 나타남

```
public:
    void LoadImageFile();
};
```

- 함수 선언에 매개변수 추가

```
public:
    void LoadImageFile(CArchive &ar);
};
```


PGM과 PPM 파일 형식 읽기

LoadImageFile() 함수의 내용을 편집

```
void CImageProDoc::LoadImageFile(CArchive &ar)
{
  int i, maxValue;
  CString type, buf;
  CFile *fp = ar.GetFile();
  CString fname = fp->GetFilePath();

  // 파일의 헤더 읽기
  if (strcmp(strrchr(fname, '.'), ".ppm") == 0 ||  strcmp(strrchr(fname, '.'), ".PPM") == 0 ||
    strcmp(strrchr(fname, '.'), ".PGM") == 0 ||  strcmp(strrchr(fname, '.'), ".pgm") == 0   )
  {
    ar.ReadString(type);
```

PGM과 PPM 파일 형식 읽기

```
  do {
    ar.ReadString(buf);
  } while (buf[0] == '#');
  sscanf_s(buf, "%d %d", &imageWidth, &imageHeight);

  do {
    ar.ReadString(buf);
  } while (buf[0] == '#');
  sscanf_s(buf, "%d", &maxValue);

  if (strcmp(type, "P5") == 0) depth = 1;
  else depth = 3;
}
```


PGM과 PPM 파일 형식 읽기

```
else if (strcmp(strrchr(fname, '.'), ".raw") == 0 ||
      strcmp(strrchr(fname, '.'), ".RAW") == 0 )
{
  if (fp->GetLength() != 256 * 256) {
    AfxMessageBox("256x256 크기의 파일만 사용가능합니다.");
    return;
  }

  imageWidth = 256;
  imageHeight = 256;
  depth = 1;
}
```

PGM과 PPM 파일 형식 읽기

```
// 기억장소 할당
 inputImg = (unsigned char **) malloc(imageHeight * sizeof(unsigned char *));
 resultImg = (unsigned char **) malloc(imageHeight * sizeof(unsigned char *));

 for (i = 0; i < imageHeight; i++) {
     inputImg[i] = (unsigned char *) malloc(imageWidth * depth);
     resultImg[i] = (unsigned char *) malloc(imageWidth * depth);
 }

 // 영상 데이터 읽기
 for (i = 0; i < imageHeight; i++)
   ar.Read(inputImg[i], imageWidth*depth);
}
```


PGM과 PPM 파일 형식 읽기

- OnDraw() 함수 수정

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

  if (pDoc->inputImg == NULL) return; // 입력 영상이 읽히지 않았으면 종료

  if (pDoc->depth == 1) { // 흑백 영상 출력
     for(int y=0; y < pDoc->imageHeight; y++)      // 입력 영상 출력
        for(int x=0; x < pDoc->imageWidth; x++)
           pDC->SetPixel(x, y, RGB(pDoc->inputImg[y][x],
                 pDoc->inputImg[y][x], pDoc->inputImg[y][x]));
```

PGM과 PPM 파일 형식 읽기

```
if (viewMode == THREE_IMAGES) {
   for(int y=0; y< pDoc->imageHeight; y++)   // 두번째 입력 영상 출력
     for(int x=0; x< pDoc->imageWidth; x++)
       pDC->SetPixel(x+pDoc->imageWidth+30,y,
                  RGB(pDoc->inputImg2[y][x],
                     pDoc->inputImg2[y][x],
                     pDoc->inputImg2[y][x]));
   for(int y=0; y< pDoc->imageHeight; y++)      // 결과 영상 출력
     for(int x=0; x< pDoc->imageWidth; x++)
       pDC->SetPixel(x+pDoc->imageWidth*2+60,y,
                  RGB(pDoc->resultImg[y][x],
                     pDoc->resultImg[y][x],
                     pDoc->resultImg[y][x]));
}
```


PGM과 PPM 파일 형식 읽기

```
    else {
       for(int y=0; y< pDoc->imageHeight; y++)      // 결과 영상 출력
         for(int x=0; x< pDoc->imageWidth; x++)
           pDC->SetPixel(x+pDoc->imageWidth+30,y,
                  RGB(pDoc->resultImg[y][x],
                     pDoc->resultImg[y][x],
                     pDoc->resultImg[y][x]));
    }
  }
  else if (pDoc->depth == 3) { // 컬러 영상 출력
    for(int y=0; y < pDoc->imageHeight; y++)      // 입력 영상 출력
      for(int x=0; x < pDoc->imageWidth; x++)
        pDC->SetPixel(x, y, RGB(pDoc->inputImg[y][3*x],
          pDoc->inputImg[y][3*x+1], pDoc->inputImg[y][3*x+2]));
```

PGM과 PPM 파일 형식 읽기

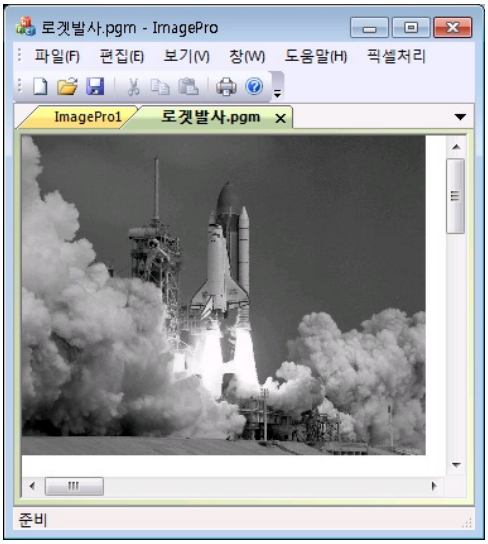
```
if (viewMode == THREE_IMAGES) {
   for(int y=0; y< pDoc->imageHeight; y++)   // 두번째 입력 영상 출력
     for(int x=0; x< pDoc->imageWidth; x++)
        pDC->SetPixel(x+pDoc->imageWidth+30,y,
            RGB(pDoc->inputImg2[y][3*x],
               pDoc->inputImg2[y][3*x+1],
               pDoc->inputImg2[y][3*x+2]));
   for(int y=0; y< pDoc->imageHeight; y++)      // 결과 영상 출력
     for(int x=0; x< pDoc->imageWidth; x++)
        pDC->SetPixel(x+pDoc->imageWidth*2+60,y,
            RGB(pDoc->resultImg[y][3*x],
               pDoc->resultImg[y][3*x+1],
               pDoc->resultImg[y][3*x+2]));
}
```


PGM과 PPM 파일 형식 읽기

```
    else {
       for(int y=0; y< pDoc->imageHeight; y++)      // 결과 영상 출력
         for(int x=0; x< pDoc->imageWidth; x++)
            pDC->SetPixel(x+pDoc->imageWidth+30,y,
               RGB(pDoc->resultImg[y][3*x],
                  pDoc->resultImg[y][3*x+1],
                  pDoc->resultImg[y][3*x+2]));
    }
  }
}
```

PGM과 PPM 파일 형식 읽기

- 프로그램을 컴파일하고 실행한 다음에 “로켓발사.pgm" 파일 열기 실행



임의의 크기 영상에 대한 픽셀 단위 처리 실습

산술 덧셈 연산 수정

OnPixelAdd() 함수 수정

- 입력 영상을 읽어 들였는 지 검사할 필요가 있음

```
void CImageProView::OnPixelAdd(void)
{
   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

   if (pDoc->inputImg == NULL) return; // 추가된 문장: 입력 영상이 있는지 검사

   pDoc->PixelAdd();     // 문장 4 : 문서 객체의 PointAdd() 함수 호출
   viewMode = TWO_IMAGES; // 문장 5 : 연산의 종류 설정
   Invalidate(FALSE);     // 문장 6 : 화면을 다시 그림
}
```


산술 덧셈 연산 수정

PixelAdd() 함수 수정

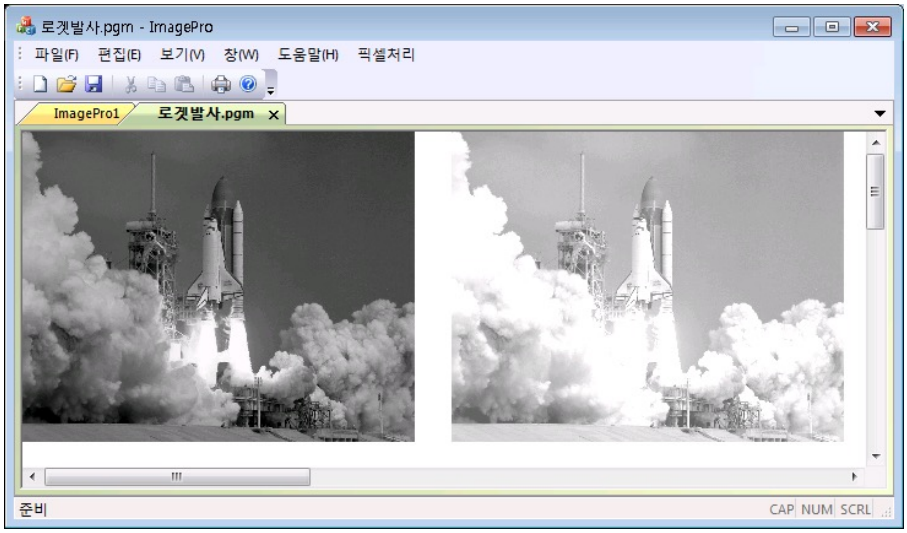
- 영상의 크기 수정

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

   for(int y=0; y < imageHeight; y++)
      for(int x=0; x < imageWidth * depth; x++) {
         value = inputImg[y][x]+100;
         if(value > 255) resultImg[y][x] = 255;
         else resultImg[y][x]=value;
      }
}
```

산술 덧셈 연산 수정

- “로켓발사.pgm” 파일을 열어서 산술 덧셈 연산 실행



두영상의 산술 덧셈 수정

- 두 영상을 읽어들이는 함수 LoadTwoImages() 함수 수정

```
void CImageProDoc::LoadTwoImages(void)
{
 CFile file;
 CFileDialog dlg(TRUE);

 AfxMessageBox("Select the First Image");
 if(dlg.DoModal()==IDOK) {
    file.Open(dlg.GetPathName(), CFile::modeRead);
    CArchive ar(&file, CArchive::load);
    LoadImageFile(ar);
    file.Close();
 }
```

두영상의 산술 덧셈 수정

```
AfxMessageBox("Select the Second Image");
if(dlg.DoModal()==IDOK) {
  file.Open(dlg.GetPathName(), CFile::modeRead);
  CArchive ar(&file, CArchive::load);
  LoadSecondImageFile(ar);
  file.Close();
 }
}
```


두영상의 산술 덧셈 수정

- CImageProDoc 클래스에 LoadSecondImageFile() 함수 추가

- 반환 형식: **void**
- 함수 이름: **LoadSecondImageFile**
- 매개변수형식 : **CArchive&**
- 매개변수이름 : **ar**

두영상의 산술 덧셈 수정

- LoadSecondImageFile() 함수을 다음과 같이 편집

```
void CImageProDoc::LoadSecondImageFile(CArchive &ar)
{
   int i, maxValue;
   CString type, buf;
   CFile *fp = ar.GetFile();
   CString fname = fp->GetFilePath();


   // 파일의 헤더 읽기
   if (strcmp(strrchr(fname, '.'), ".ppm") == 0 ||
      strcmp(strrchr(fname, '.'), ".PPM") == 0 ||
      strcmp(strrchr(fname, '.'), ".PGM") == 0 ||
      strcmp(strrchr(fname, '.'), ".pgm") == 0 )
   {
      ar.ReadString(type);
```


두영상의 산술 덧셈 수정

```
  do {
     ar.ReadString(buf);
  } while (buf[0] == '#');
  sscanf_s(buf, "%d %d", &imageWidth, &imageHeight);

  do {
     ar.ReadString(buf);
  } while (buf[0] == '#');
  sscanf_s(buf, "%d", &maxValue);

  if (strcmp(type, "P5") == 0) depth = 1;
  else depth = 3;
}
else if (strcmp(strrchr(fname, '.'), ".raw") == 0 ||
         strcmp(strrchr(fname, '.'), ".RAW") == 0 )
{
   if (fp->GetLength() != 256 * 256) {
      AfxMessageBox("256x256 크기의 파일만 사용가능합니다.");
      return;
   }
```

두영상의 산술 덧셈 수정

```
    imageWidth = 256;
    imageHeight = 256;
    depth = 1;
  }

  // 기억장소 할당
  inputImg2 = (unsigned char **) malloc(imageHeight * sizeof(unsigned char *));

  for (i = 0; i < imageHeight; i++) {
    inputImg2[i] = (unsigned char *) malloc(imageWidth * depth);
  }

  // 영상 데이터 읽기
  for (i = 0; i < imageHeight; i++)
    ar.Read(inputImg2[i], imageWidth*depth);
}
```


두영상의 산술 덧셈 수정

- PixelTwoImageAdd() 함수의 내용을 다음과 같이 수정

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

   LoadTwoImages();

   for(int y=0; y<imageHeight; y++)
      for(int x=0; x < imageWidth * depth; x++) {
         value = inputImg[y][x] + inputImg2[y][x];
         if (value > 255) resultImg[y][x] = 255;
         else resultImg[y][x] = value;
      }
}
```

두영상의 산술 덧셈 수정

- “모나리자.pgm" 파일과 "모나리자-mask.pgm" 파일을 이용하여 두 영상의 산술 덧셈 연산을 수행하여 그림과 같은 결과가 나타나는지 확인

영상의 선명화 실습

영상의 선명화

- 메뉴 막대에 [영역 처리] 메뉴 추가

- 부메뉴 추가
  - 이름 : 선명화
  - ID : ID_REGION_SHARPENING

영상의 선명화

- 메뉴에 대한 이벤트 처리기를 OnRegionSharpening() 함수로 추가하고 다음과 같이 편집

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

    if (pDoc->inputImg == NULL) return;
    pDoc->RegionSharpening();
    viewMode = TWO_IMAGES;
    Invalidate(FALSE);
}
```

영상의 선명화

- CImageProDoc 클래스에 RegionSharpening() 함수 추가
  - 반환 형식: void
  - 함수 이름: RegionSharpening

영상의 선명화

- RegionSharpening() 함수의 내용을 다음과 같이 편집

```
void CImageProDoc::RegionSharpening(void)
{
  float kernel[3][3] = {{0, -1, 0}, {-1, 5, -1}, {0, -1, 0}};


  Convolve(inputImg, resultImg, imageWidth, imageHeight, kernel, 0, depth);
}
```

영상의 선명화

- CImageProDoc 클래스에 Convolve() 함수 추가
  - 반환 형식: void
  - 함수 이름: Convolve
  - 매개변수

번호	매개 변수 형식	매개 변수 이름
1	unsigned char **	inputImg
2	unsigned char **	resultImg
3	int	cols
4	int	rows
5	float [][3]	mask
6	int	bias
7	int	depth

영상의 선명화

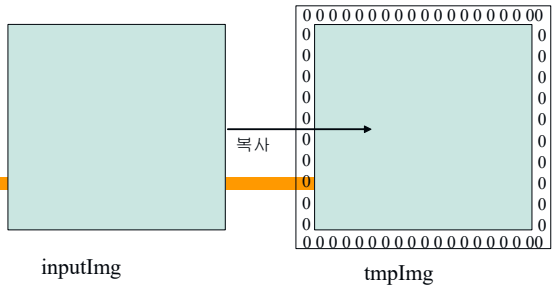
- Convolve() 함수 편집

```
void CImageProDoc::Convolve(unsigned char **inputImg,
   unsigned char **resultImg, int cols, int rows, float mask[][3], int bias, int depth)
{
   int i, j, x, y;
   int red, green, blue;
   int sum;
   unsigned char **tmpImg;

   // 기억장소 할당
   tmpImg = (unsigned char **) malloc((imageHeight + 2)* sizeof(unsigned char *))

   for (i = 0; i < imageHeight + 2; i++)
      tmpImg[i] = (unsigned char *) malloc((imageWidth + 2) * depth);
```

영상의 선명화




```
// 0-삽입을 위해 0으로 초기화
   for (y = 0; y < imageHeight + 2; y++)
       for (x = 0; x < (imageWidth + 2) * depth; x++)
          tmpImg[y][x] = 0;

// 영상 복사
   for (y = 1; y < imageHeight + 1; y++)
      for (x = 1; x < imageWidth + 1; x++)
        if (depth == 1) tmpImg[y][x] = inputImg[y-1][x-1];
        else if (depth == 3) {
          tmpImg[y][3*x] = inputImg[y-1][3*(x-1)];
          tmpImg[y][3*x+1] = inputImg[y-1][3*(x-1)+1];
          tmpImg[y][3*x+2] = inputImg[y-1][3*(x-1)+2];
        }
```


영상의 선명화

```
for (y=0; y < imageHeight; y++)
      for (x=0; x < imageWidth; x++) {
         if (depth == 1) {
            sum=0;
            for (i=0; i<3; i++)
               for (j=0; j<3; j++)
                  sum += (int) (tmpImg[y+i][x+j] * mask[i][j]) ;

             sum = sum + bias;

            if (sum > 255) sum=255;
            if (sum < 0) sum = 0;
            resultImg[y][x] = (unsigned char) sum;
         }
```

영상의 선명화

```
else if (depth == 3) {
    red = 0;
    green = 0;
    blue = 0;

    for (i=0; i<3; i++)
       for (j=0; j<3; j++) {
          red += (int) (tmpImg[y+i][3*(x+j)] * mask[i][j]);
          green += (int) (tmpImg[y+i][3*(x+j)+1] * mask[i][j]);
          blue += (int) (tmpImg[y+i][3*(x+j)+2] * mask[i][j]);
       }
     red = red + bias;
     green = green + bias;
     blue = blue + bias;
```


영상의 선명화

```
          if (red > 255) red = 255;
          if (red < 0) red = 0;
          if (green > 255) green = 255;
          if (green < 0) green = 0;
          if (blue > 255) blue = 255;
          if (blue < 0) blue = 0;

          resultImg[y][3*x] = (unsigned char) red;
          resultImg[y][3*x+1] = (unsigned char) green;
          resultImg[y][3*x+2] = (unsigned char) blue;
       }
   }

  // 기억장소 반환
  for (i = 0; i < imageHeight + 2; i++) free(tmpImg[i]);
  free(tmpImg);
}
```

영상의 선명화

- 프로그램을 컴파일하고 실행해보자. “모나리자.pgm" 파일을 열은 다음에 [선명화] 메뉴를 선택