1. **Spatial-domain Filtering**

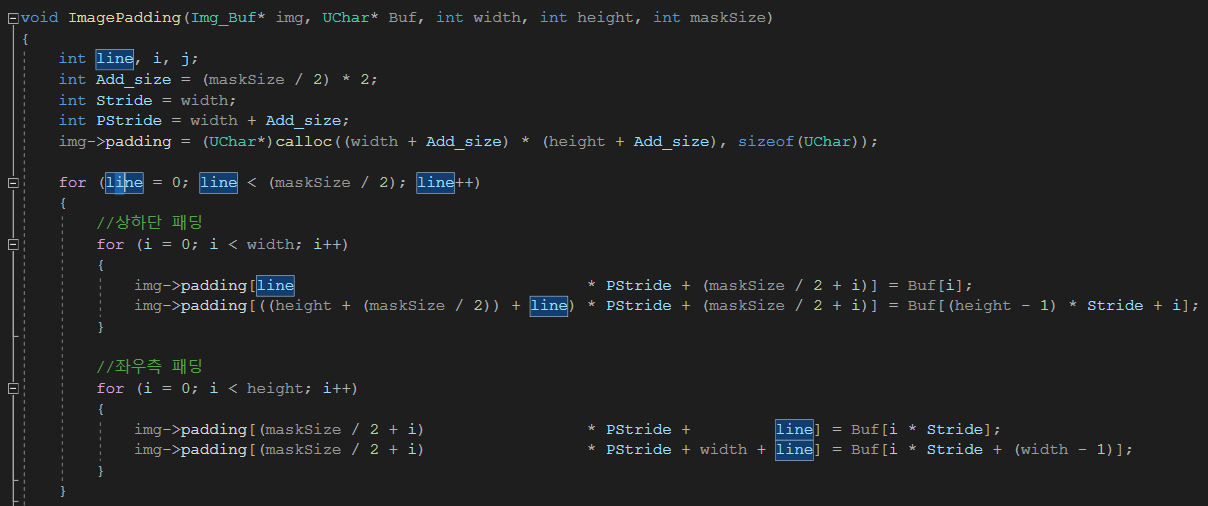
**:** Spatial-domain Filtering은 공간 영역(영상 : 2D)에서 filter를 적용하여 가운데 화소의 값을 바꾸는 영상처리 기법입니다. Spatial-domain filtering은 convolution과 관련이 있고, convolution은 convolution kernel과 관계 있습니다.

Convolution

* Convolution이란 현재 화소를 중심으로 이웃한 화소들이 포함한 영역을 회선 마스크와 곱한 후 합한 결과입니다. 이때 사용되는 영상은 padding을 적용한 영상입니다.
* Convolution kernel에는 **Smoothing kernel**과 **Differentiating kernel**이 있습니다. Smoothing kernel은 자신과 주변화소들의 평균으로, data를 부드럽게 만들고 noise효과를 줄일 수 있습니다. Differentiating kernel은 영상에서 신호가 급격하게 변하는 것을 더 빠르게 만드는 kernel로, edge를 검출할 때 사용합니다.

**실습 1) Padding**

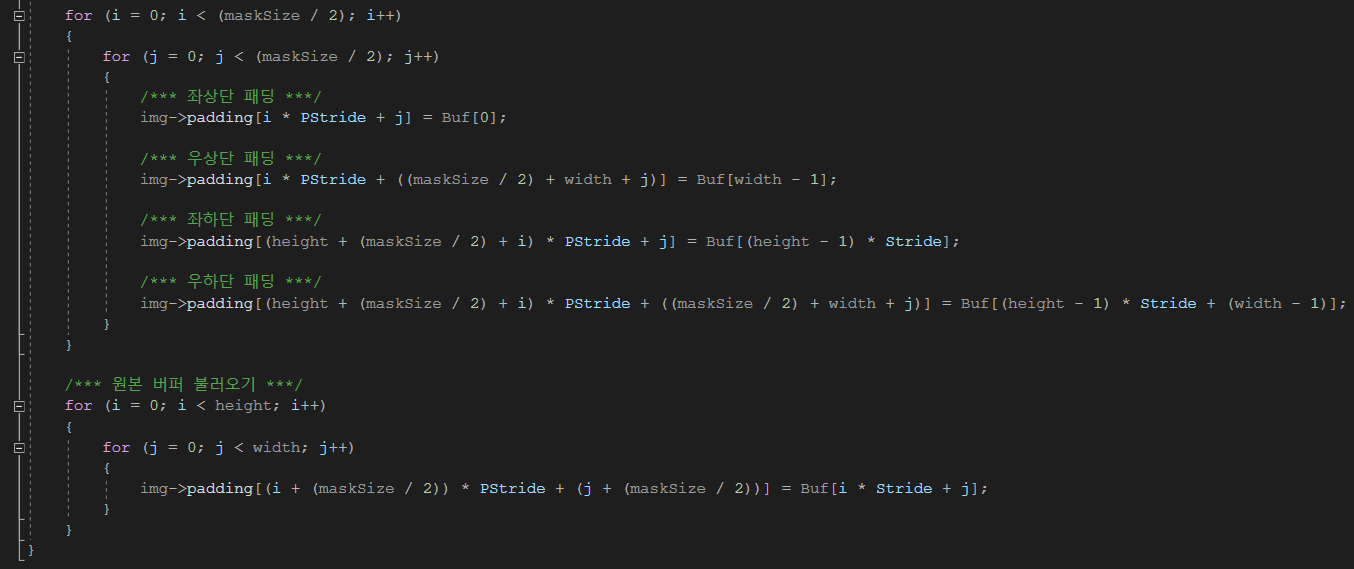
* Convolution을 수행할 때 결과 영상의 size가 작아지는 것을 방지하기 위해서 Padding기법을 사용합니다. Padding은 원본 영상 주변으로 mask의 크기에 따라 값을 새로 넣는 기법인데, 영상의 외곽부분을 0으로 채우는 **Zero padding**과 주변의 가장 인접한 값으로 채우는 **Copy padding**이 있습니다.
* Copy Padding



* Padding을 수행할 때 추가되는 화소의 수는 mask의 크기에 따라 결정됩니다. 예를 들어 3x3 filter를 4x4 영상에 적용할 때에는 padding을 적용하지 않고 convolution을 수행하게 되면 결과 영상은 2x2 크기로 출력됩니다. 따라서 3x3 filter를 적용할 때에는 상하좌우 한 줄 씩, 5x5 filter를 적용할 때에는 두 줄 씩, 7x7 filter를 적용할 때는 세 줄 씩 화소의 값을 새로 넣어주어야 합니다. 새로 넣어주는 화소의 수는 Add\_size 변수에 선언되어 있습니다. 예를 들어 maskSize가 3x3으로 3이라면, Add\_size의 maskSize/2\*2 대응되어 총 2개의 pixel이 추가됩니다. 이는 위아래(height) 하나씩 총 2개, 왼쪽/오른쪽(width) 하나씩 총 2개를 뜻합니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 7 | 5 | 4 |  |
| 7 | 7 | 5 | 4 | 4 |
| 2 | 2 |  | 8 | 8 |
| 6 | 6 | 3 | 1 | 1 |
|  | 6 | 3 | 1 |  |

* Padding된 영상의 size의 크기를 원본 영상의 width와 height에 Add\_size를 더하여 공간할당을 해야 합니다. Copy padding은 주변의 가장 인접한 값으로 화소를 추가하기 때문에 먼저 영상의 상하좌우 가장자리에 가장 인접한 화소를 하나씩 추가합니다. 예시로, 왼쪽 그림은 3x3 원본 영상(초록색)에 한 줄 padding을 적용한 것입니다. 노란색 부분은 padding을 수행한 화소로, 가장 인접한 화소를 추가합니다.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 7 | 5 | 4 | 4 |
| 7 | 7 | 5 | 4 | 4 |
| 2 | 2 |  | 8 | 8 |
| 6 | 6 | 3 | 1 | 1 |
| 6 | 6 | 3 | 1 | 1 |

* Padding은 영상의 겉 부분을 감싸는 것이기 때문에 상하좌우 각 모서리 또한 가장 인접한 화소로 추가해야 합니다. 예시로 왼쪽 그림에서 각 모서리인 빨간색 화소 부분 또한 가장 인접한 화소로 추가해야 합니다.

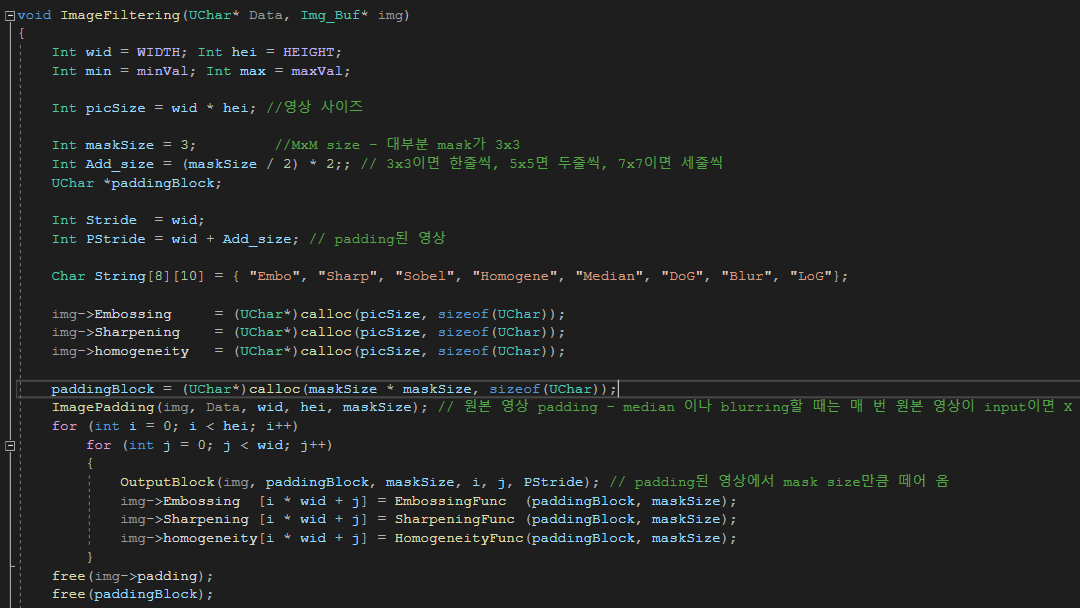
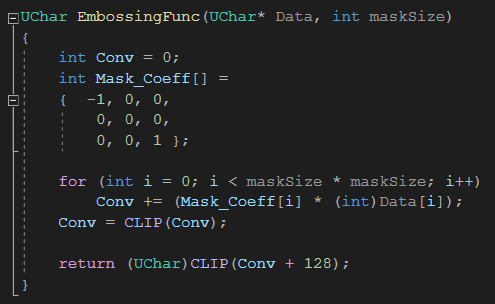
**실습 2) Filtering**

1. **엠보싱**

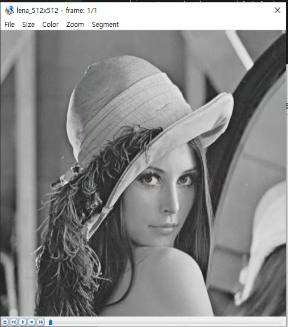
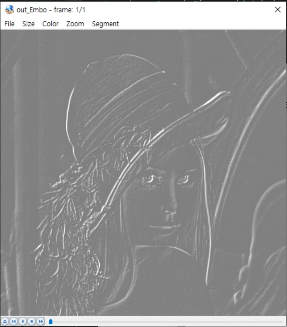
* Embossing기법은 nickel판에 etching효과를 주는 mask입니다.

|  |  |  |
| --- | --- | --- |
| -1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 1 |

* Embossing mask는 왼쪽과 같으며, 중앙에 위치한 계수에 대하여 다른 계수들을 상쇄시키는 효과가 있습니다.
* Center weight가 0이고, 합계가 0이기 때문에 영상의 모양은 거의 날라간다는 특징이 있습니다.
* 이 때 clipping은 2번 수행해야 합니다. Convolution을 수행하여 새로운 pixel값을 구할 때 한 번, 앞에서 구한 convolution 결과값에 128을 더하여 최종적인 출력 pixel을 구할 때 한 번, 총 2번 clipping을 수행합니다.
* Embossing을 수행할 때 input 영상은 원본 lena입니다.

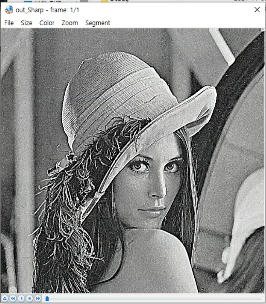
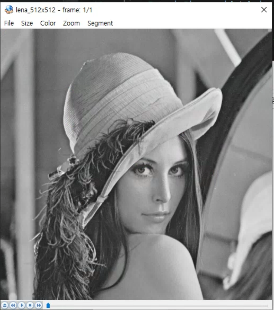
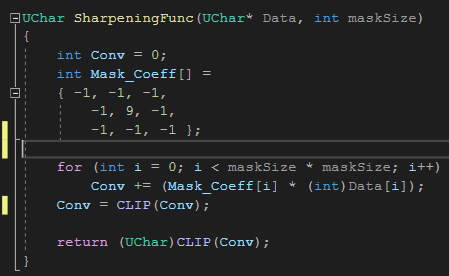
* + - * ImageFiltering함수에서 Embossing을 수행합니다. 영상의 크기만큼 동적할당 한 후, 반복문을 통해 원본 영상에 padding을 적용합니다. Padding이 적용된 영상의 3x3 블록을 떼어와서 EmbossingFunc에서 embossing mask로 convolution을 수행합니다. Convolution을 수행할 때, convolution이 수행된 영상에 128을 더할 때 총 두 번 clipping을 수행합니다.

* + - * 원본 영상과 embossing이 적용된 영상입니다. embossing filter로 convolution을 수행하고 128을 더하여, embossing이 적용된 영상은 edge부분에 흰 색으로 색상이 존재하고, 신호의 변화가 크게 급격하지 않은 부분에서는 회색을 띠는 것을 알 수 있습니다.

1. **샤프닝**

* Sharpening은 high-pass filter로, 영상에서 상세한 부분들(고주파 부분)을 더욱 강조하는 효과가 있습니다.
* Sharpening을 수행할 때 input 영상은 원본 lena입니다.



* + - * ImageFiltering함수에서 sharpening을 수행합니다. 영상의 크기만큼 동적할당 한 후, 반복문을 통해 원본 영상에 padding을 적용합니다. Padding이 적용된 영상의 3x3 블록을 떼어와서 SharpeningFunc에서 sharpening mask로 convolution을 수행합니다. Padding된 영상에서 3x3 블록을 떼서 각 자리 곱하고, 더하는 과정에서 clipping을 수행합니다. (1번)

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| -1 | 9 | -1 |
| -1 | -1 | -1 |

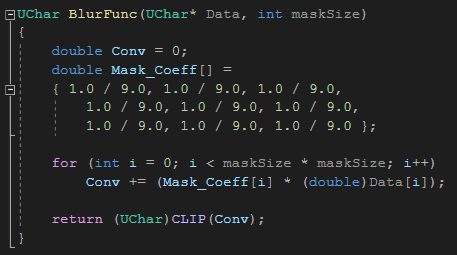
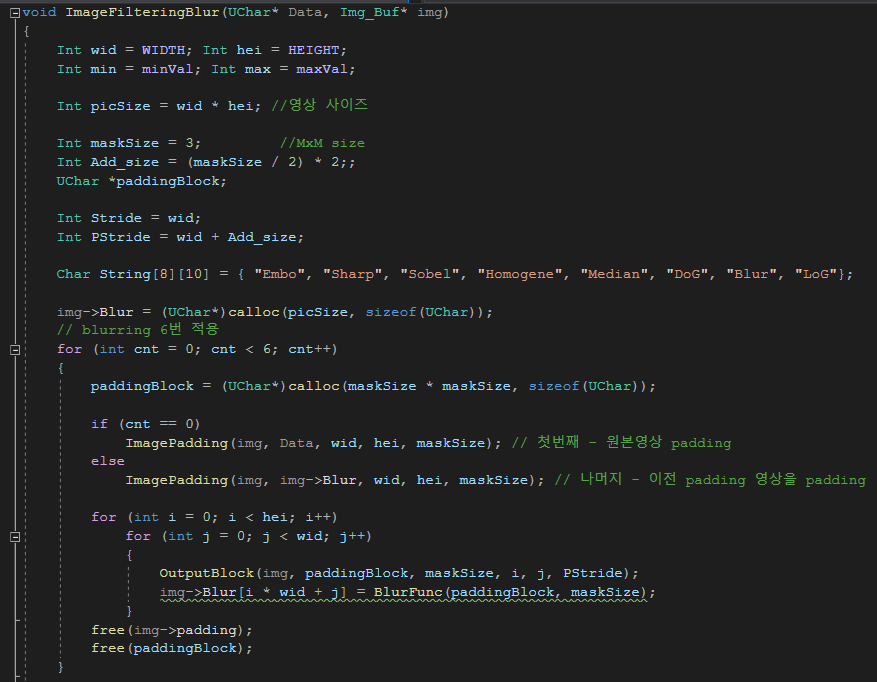
* + - * Sharpening mask는 왼쪽과 같습니다. 이를 sharpeningFunc의 Mask\_Coeff 배열에 넣어줍니다.
      * 결과 영상을 비교해보면, 영상에서 상세한 부분들이 강조되었음을 알 수 있습니다.

1. **블러링**

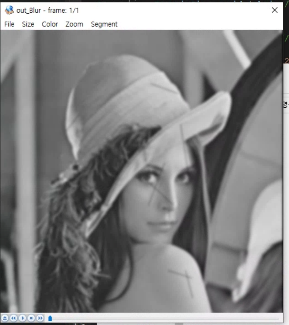
* Blurring은 영상의 noise를 제거하는 효과가 있습니다. 영상을 흐리게 하거나, 배경을 약화시키고자 할 때 사용하는 영상처리 기법입니다.

|  |  |  |
| --- | --- | --- |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

* Blurring mask는 왼쪽과 같으며, 현재 화소를 이웃 화소들과 평균한 결과입니다.
* Blurring을 수행 할 때 input 영상은 scratch lena입니다.



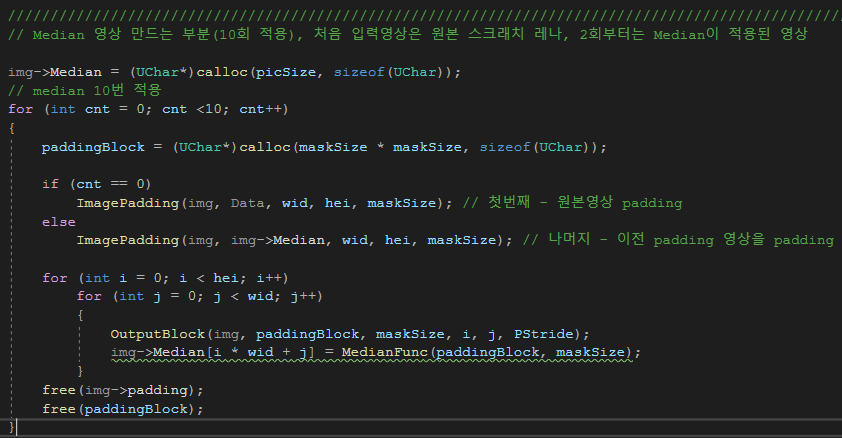
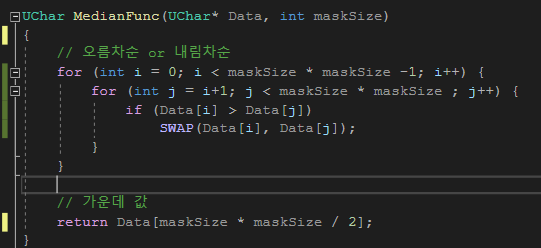
* + - * ImageFilteringBlur함수에서 blurring을 수행합니다. 영상의 크기만큼 동적할당 한 후, 반복문을 통해 원본 영상에 padding을 적용합니다. Padding이 적용된 영상의 3x3 블록을 떼어와서 BlurFunc에서 Bluring mask로 convolution을 수행합니다. Padding된 영상에서 3x3 블록을 떼서 각 자리 곱하고, 더하는 과정에서 clipping을 수행합니다. (1번)
      * 블러링을 6번 수행할 때마다 padding 또한 6번 수행해야 합니다. 이를 위해 ImageFilteringBlur 함수에서 padding을 적용하는 반복문을 6번으로 지정합니다. 처음 블러링을 수행할 때는 원본 영상을 padding해야하지만, 그 다음 블러링을 수행할 때는 이전에 padding한 영상을 다시 padding 해야 합니다.

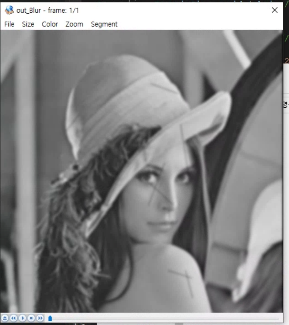
* + - * 결과 영상을 비교해 보면, scratch lena 영상의 scratch가 미세하게 지워지지만 강한 noise라서 아예 지워지지는 않습니다. 또한 영상이 전체적으로 흐려진다는 것을 알 수 있습니다.

1. **미디언 필터링**

* median filtering은 Nonlinear filters로, convolution을 활용하지 않는 filter입니다. 영상에 존재하는 noise가 Gaussian 분포인 Gaussian noise일 때는 Gaussian smoothing을 사용하면 효과적으로 noise를 줄일 수 있지만, noise가 Gaussian noise가 아닌 굉장히 큰 값일 때는 주변으로 전파되어 화질이 낮아집니다. 즉, 영상신호에 굉장히 큰 noise가 있을 때 Gaussian smoothing 기법을 사용하는 것은 적합하지 않기 때문에, 이 때 median filter를 사용합니다.
* Median filtering은 filter window 내에 있는 화소 값들을 오름차순 또는 내림차순 하여 그 중간값(median)을 새로운 화소로 사용하는 방법입니다. Filter window 내에 있는 화소들은 굉장히 가깝게 있는 화소이므로, pixel value가 크게 차이 나지 않습니다. 따라서 noise가 있었을 때 주변 화소 값으로 효과적으로 제거할 수 있습니다.
* Median filtering을 수행할 때 input 영상은 scratch lena입니다.

* + - * ImageFilteringBlur함수에서 median filtering을 수행합니다. 영상의 크기만큼 동적할당 한 후, 반복문을 통해 원본 영상에 padding을 적용합니다. Padding이 적용된 영상의 3x3 블록을 떼어와서 MedianFunc에서 떼어 온 블록의 화소 값들을 bubble sort로 오름차순하고, 가운데 값을 반환합니다. 단순히 중간 화소 값을 그대로 가져오므로 clipping과정을 수행하지 않아도 됩니다.
      * Median filtering을 10번 수행할 때마다 padding 또한 10번 수행해야 합니다. 이를 위해 ImageFilteringBlur 함수에서 padding을 적용하는 반복문을 10번으로 지정합니다. 처음 median filtering을 수행할 때는 원본 영상을 padding해야하지만, 그 다음 median filtering을 수행할 때는 이전에 padding한 영상을 다시 padding 해야 합니다.

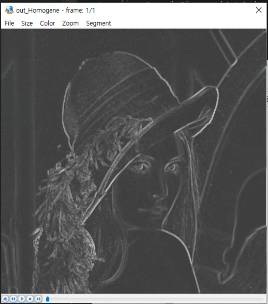
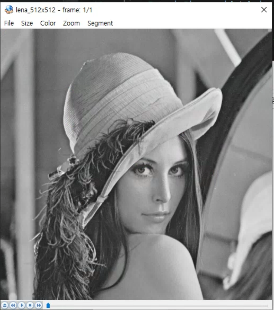
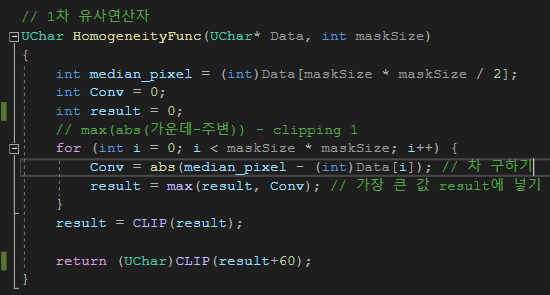
* + - * 결과 영상을 비교해보면, blurring을 수행했을 때(오른쪽)보다 더욱 효과적으로 noise가 제거되었고, 영상의 전체적인 흐림 정도도 적다는 것을 알 수 있습니다. Scratch lena(왼쪽)의 noise 정도가 굉장히 강하기 때문에 blurring보다 median filtering(가운데)이 noise를 제거하는데 효과적임을 확인할 수 있습니다.

**실습 2) Edge Detection**

* Edge detection은 영상의 object의 윤곽선을 감지하는 기술로, high-pass filter 입니다. High-pass filter는 높은 주파수 성분을 통과시키는 filter로, 주파수가 높다는 것은 영상 신호의 밝기가 급격하게 변화하는 edge를 뜻합니다.

1. **유사 연산자(Homogeneity operator)**

* 유사연산자는 edge detection에 사용되는 연산자로, 가운데 pixel과 그 주위를 감싸고 있는 pixel들의 뺀 값을 절대값을 취해, 그 중 최대값을 new pixel로 사용하는 연산자입니다. 결과 영상을 통해 모든 화소 위치에서 밝기의 변화가 어느 정도 있는지 알 수 있기 때문에, 크면 클수록 밝기 변화가 급격한 화소라는 것을 짐작할 수 있습니다.
* 유사 연산자는 2번의 clipping과정을 거칩니다. 첫 번째 clipping은 주변 화소들과 가운데 화소의 뺄셈 중 maximum 값을 얻어낸 후 그 maximum 결과값을 clipping하고, 조금 더 선명하게 보기 위해 clipping된 maximum 값에 60을 더할 때 한 번 더 clipping을 수행합니다.
* 유사 연산자를 수행할 때 input 영상은 원본 lena입니다.



* + - * ImageFiltering함수에서 유사 연산자를 수행합니다. 영상의 크기만큼 동적할당 한 후, 반복문을 통해 원본 영상에 padding을 적용합니다. Padding이 적용된 영상의 3x3 블록을 떼어와서 HomogeneityFunc에서 가운데 화소 값과 그 화소를 둘러싸고 있는 pixel들의 차의 절대값을 구한 후, 그 중 최대값을 가지는 화소 값을 clipping합니다. 최대값 화소에 clipping을 적용한 값에 60을 더할 때 clipping을 한 번 더 수행합니다.
      * 유사 연산자를 수행했을 때 원본 영상의 edge를 잘 검출하는 것을 결과 영상을 통해 확인할 수 있습니다.

1. **1차 미분 연산자**

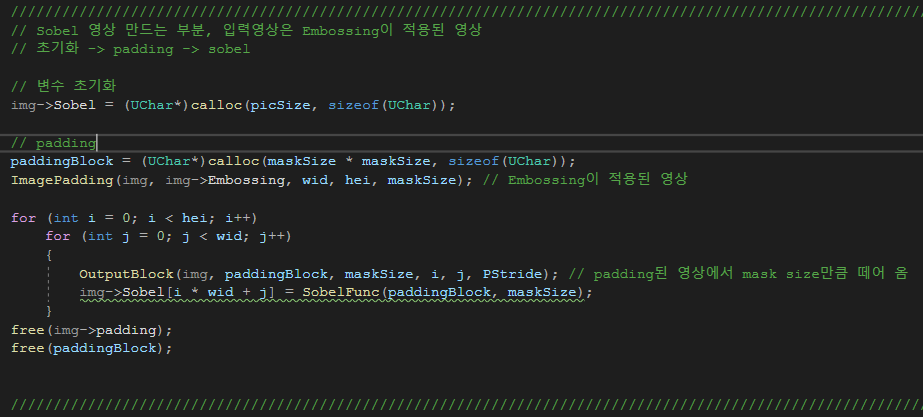
* 1차 미분 연산자를 2차원 영상신호로 확장한 것을 ‘**Image Gradient**’라고 합니다. Image Gradient는 변화를 감지할 때 x방향과 y방향의 변화를 따로 분리해서 감지한다는 특징이 있습니다. Image Gradient를 수행할 때 영상의 noise를 줄이기 위해 smoothing을 먼저 적용하고, gradient를 계산합니다.
* 1차 미분 연산자는 Gaussian kernel을 어떤 것을 사용하고, 미분을 구하는 operator가 무엇인지에 따라서 구분할 수 있습니다. 종류에는 ‘Prewitt Operator’, ‘Sobel Operator’, ‘Scharr Operator’, ‘Roberts Cross Operator’ 등이 있습니다.
* **Sobel Operator**
  + Sobel Operator는 smoothing kernel로 분산이 0.5인 Gaussian 분포를 사용합니다.

|  |  |  |
| --- | --- | --- |
| 1 | 0 | -1 |
| 2 | 0 | -2 |
| 1 | 0 | -1 |

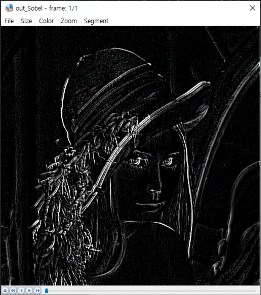
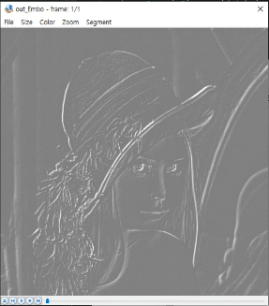
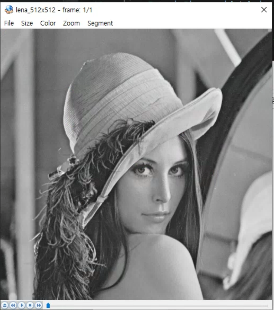
* + Sobel Operator는 수평, 수직 방향의 기울기를 검출하고, 수평과 수직 윤곽선보다 대각선 방향의 윤곽선에 더욱 민감하다는 특징이 있습니다. 또한 kernel이 central을 중심으로 상하좌우의 weight가 좀 더 크다는 점에는 noise에 더 robust합니다. 왼쪽에 있는 첫번째 표가 Sobel Row mask이고, 두번째 표가 Sobel Col 마스크입니다.

|  |  |  |
| --- | --- | --- |
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

* + Sobel Operator는 수평과 수직 방향으로 각각 convolution을 수행하고, 그 값을 더하여 새로운 pixel 값을 만든다는 점에서 clipping이 3번 수행되어야 합니다.
  + 입력 영상은 embossing이 적용된 lena 영상입니다.



* + - * ImageFiltering함수에서 sobel operator를 수행합니다. 영상의 크기만큼 동적할당 한 후, 반복문을 통해 원본 영상에 padding을 적용합니다. Padding이 적용된 영상의 3x3 블록을 떼어와서 SobelFunc에서 Row mask를 뜻하는 Mask\_Coeff\_Ver, Col mask를 뜻하는 Mask\_Coeff\_Hor를 적용하여 수평과 수직 방향으로 각각 convolution을 수행합니다. 이 때 Row방향과 Col방향을 각각 수행하고, 두 과정을 각각 clipping합니다(clipping 2번). Clipping된 수평과 수직 방향의 값을 더하는 과정에서도 clipping을 수행하고 반환합니다(clipping 총 3번).



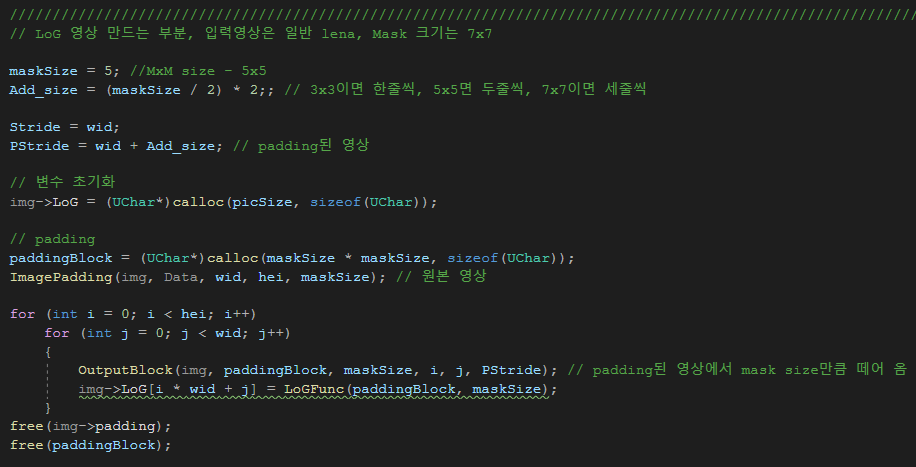
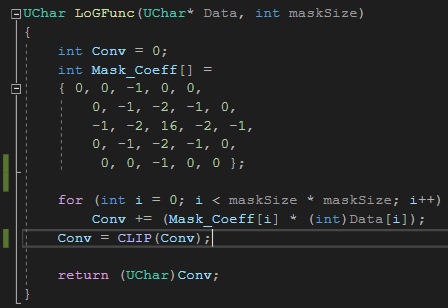
* + - * 원본 영상(왼쪽)에서 embossing을 적용(가운데)한 후 Sobel operator를 적용(오른쪽)한 결과 영상입니다. Sobel operator를 적용했을 때도 edge가 검출되었다는 것을 확인할 수 있습니다.

1. **2차 미분 연산자**

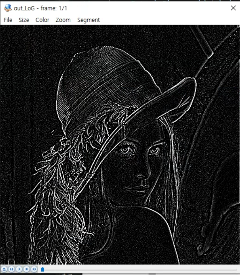
* 2차 미분 연산자는 1차 미분을 두 번 수행하는 연산자입니다. 2차 미분을 수행함으로써 edge가 있는 곳에서 zero crossing(신호의 부호 변화를 발생시키는 점)이 발생함을 알 수 있고, zero crossing point를 찾으면 edge의 모양을 얇게 구하는데 도움이 됩니다. 하지만 zero crossing이 너무 많으면, 오히려 edge검출에 불리할 수도 있다는 특징이 있습니다.
* 2차 미분 연산자에는 ‘Laplacian of Gaussian(LoG)’, ‘Difference of Gaussians(DoG)’이 있습니다.
* **LoG (Laplaclan of Gaussian)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | -1 | 0 | 0 |
| 0 | -1 | -2 | -1 | 0 |
| -1 | -2 | 16 | -2 | -1 |
| 0 | -1 | -2 | -1 | 0 |
| 0 | 0 | -1 | 0 | 0 |

* + 왼쪽은 LoG의 mask입니다. LoG의 mask가 5x5이기 때문에 padding 또한 3x3 mask와 달리 상하좌우로 두 줄 씩 추가됩니다.
  + 입력 영상은 원본 lena 영상입니다.

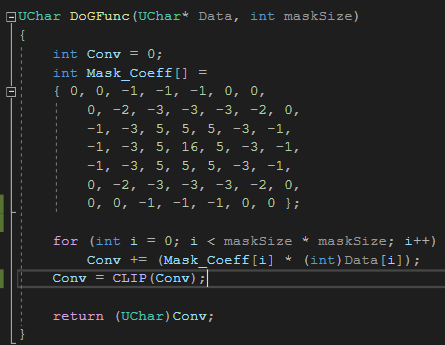
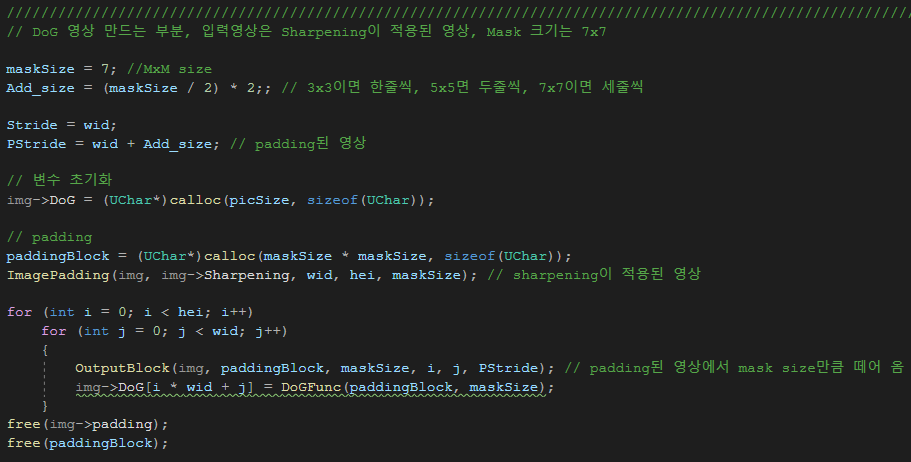
* + - * ImageFiltering함수에서 LoG operator를 수행합니다. 영상의 크기만큼 동적할당 한 후, 반복문을 통해 원본 영상에 padding을 적용합니다. 위 실습과 달리 LoG mask가 5x5이기 때문에 maskSize를 5로 갱신해야 합니다. maskSize가 5임에 따라 Add\_Size는 4로 갱신되고, 상하좌우 각각 2줄씩 padding 됩니다. Padding이 적용된 영상의 5x5 블록을 떼어와서 LoGFunc에서 convolution을 수행합니다. Convolution을 수행한 pixel에 clipping을 적용하여 반환한 값을 new pixel로 사용합니다.

  :: Log를 수행했을 때에도 edge가 검출됩니다.

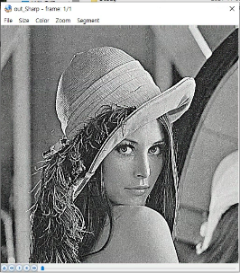
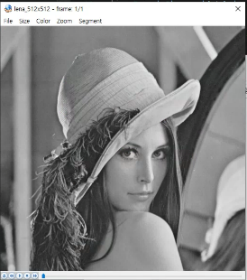
* **DoG (Difference of Gaussians)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | -1 | -1 | -1 | 0 | 0 |
| 0 | -2 | -3 | -3 | -3 | -2 | 0 |
| -1 | -3 | 5 | 5 | 5 | -3 | -1 |
| -1 | -3 | 5 | 16 | 5 | -3 | -1 |
| -1 | -3 | 5 | 5 | 5 | -3 | -1 |
| 0 | -2 | -3 | -3 | -3 | -2 | 0 |
| 0 | 0 | -1 | -1 | -1 | 0 | 0 |

* + LoG kernel은 분산이 커짐에 따라 커지고, 이에 따라 계산양이 증가합니다. 따라서 LoG kernel을 대체할 수 있는 조금 더 작은 size의 kernel이 DoG입니다
  + DoG operator를 사용하면 kernel의 size를 작게 유지하여 계산양을 줄이면서 LoG kernel의 효과를 나타내는 kernel을 만들어낼 수 있습니다.
  + DoG mask는 왼쪽과 같이 7x7입니다. 따라서 상하좌우로 세 줄 씩 padding됩니다. 입력 영상은 sharpening이 적용된 lena 영상입니다.



* + - * ImageFiltering함수에서 DoG operator를 수행합니다. 영상의 크기만큼 동적할당 한 후, 반복문을 통해 원본 영상에 padding을 적용합니다. 위 실습과 달리 DoG mask가 7x7이기 때문에 maskSize를 7로 갱신해야 합니다. maskSize가 7임에 따라 Add\_Size가 6으로 갱신되고, 상하좌우 각각 3줄씩 padding 됩니다. Padding이 적용된 영상의 7x7 블록을 떼어와서 DoGFunc에서 convolution을 수행합니다. Convolution을 수행한 pixel에 clipping을 적용하여 반환한 값을 new pixel로 사용합니다.



* + - * 원본 영상(왼쪽)에서 sharpening을 적용(가운데)한 후 DoG를 적용(오른쪽)한 결과 영상입니다. 영상에 Sharpening이 적용되어 영상에서 상세한 부분들이 더욱 강조되었기 때문에, DoG를 적용했을 때 edge와 강조된 부분들이 모두 검출되었다는 것을 알 수 있습니다.