컴퓨터그래픽스 3주차 과제 보고서
201300995 이상건 물리학과

구현 내용
Laplacian of Gaussian 과 Derivative of Gaussian을 이용해서 edge를 찾아내는 내용이었다.

이유(구현 방법)
우선 이론을 그대로 따라하기로 했다.
LoG의 경우 2차미분한 가우시안 함수를 커널로 만들어서 그대로 넣었다. 앞의 계수도 넣었다.

```
def my_LoG(img, ksize=7, boundary_=0):  # default sigma =1, sigma = 0.3(n/2 -1) + 0.8
    """
    :param img: LoG edge detection를 수행할 이미지.
    :param ksize: Kernel size. ksize x ksize의 kernel 사용.
    :param boundary: filtering 경계 처리 방법. 0 : zero-padding,(default) 1 : repetition, 2 : mirroring
    :return: LoG 방법으로 찾아낸 Edge 이미지.
    """
    sigma = 2.5
    LoG = np.ones((ksize,ksize))
    middleX = ksize//2
    middleY = ksize//2
    gesu = (1 / (2 * np.pi * np.square(sigma)))
    for i in range(ksize):
        for j in range(ksize):
            LoG[i][j] = -1 * gesu *(1-((np.square((i-middleX)) + np.square(j-middleY)) / (2*np.square(sigma)))) * (1/(np.pi * sigma ** 4)) * np.exp(-1 * (np.square((i-middleX)) + np.square(j-middleY)) / (2 * np.square(sigma)))

    LoG_img = my_filtering(img, LoG, boundary=boundary_) # LoG로 만들어진 kernel
    LoG_img = find_zerocrossing(LoG_img)
    return LoG_img
```

```python
def find_zerocrossing(LoG, thresh = 0.01): #0이 아니라 thresh를 사용하는건 실수 연산의 오차 고려.
    '''
    :param LoG: zero-crossing을 검사할 LoG 필터링 된 Image
    :param thresh: 실수 연산에서 생기는 오차가 있을 수 있기 때문에, 0이 아니라 thresh를 이용해서 수행.
                   (지우고 0으로 zero-crossing을 검사해도 괜찮습니다.)
    :return: zero-crossing 지점만 255값을 가지는 이미지.
    '''

    buho = [thresh,thresh,thresh,thresh,thresh,thresh,thresh,thresh]
    y, x = len(LoG), len(LoG[0])
    res = np.zeros((y,x), dtype = np.uint8)
    for i in range (1, y-1): #맨 처음과 맨 마지막은 제외.(검사에서 경계를 넘어가지 않도록)
        for j in range (1, x-1):
            res[i][j] = 0
            if (LoG[i][j] == 0):
                print("zero")
            # zero-crossing을 검사하는 코드를 작성해주세요.
            buho[0] = LoG[i - 1][j - 1]
            buho[1] = LoG[i - 1][j]
            buho[2] = LoG[i - 1][j + 1]
            buho[3] = LoG[i][j + 1]
            buho[4] = LoG[i + 1][j + 1]
            buho[5] = LoG[i + 1][j]
            buho[6] = LoG[i + 1][j-1]
            buho[7] = LoG[i][j - 1]
            if (buho[0] > thresh ):
                for k in range(1,8):
                    if (buho[k] < thresh):
                        res[i][j] = 225
                        break
            if (buho[0] < thresh ):
                for k in range(1,8):
                    if (buho[k] > thresh):
                        res[i][j] = 225
                        break


    return res
```

제로 크로싱의 경우 배열에 넣어서 비교하는 식으로 해결했다. 배열의 처음 원소가 +일 경우 -가 하나라도 나오면 제로크로싱이라고 판단하는 식이다.

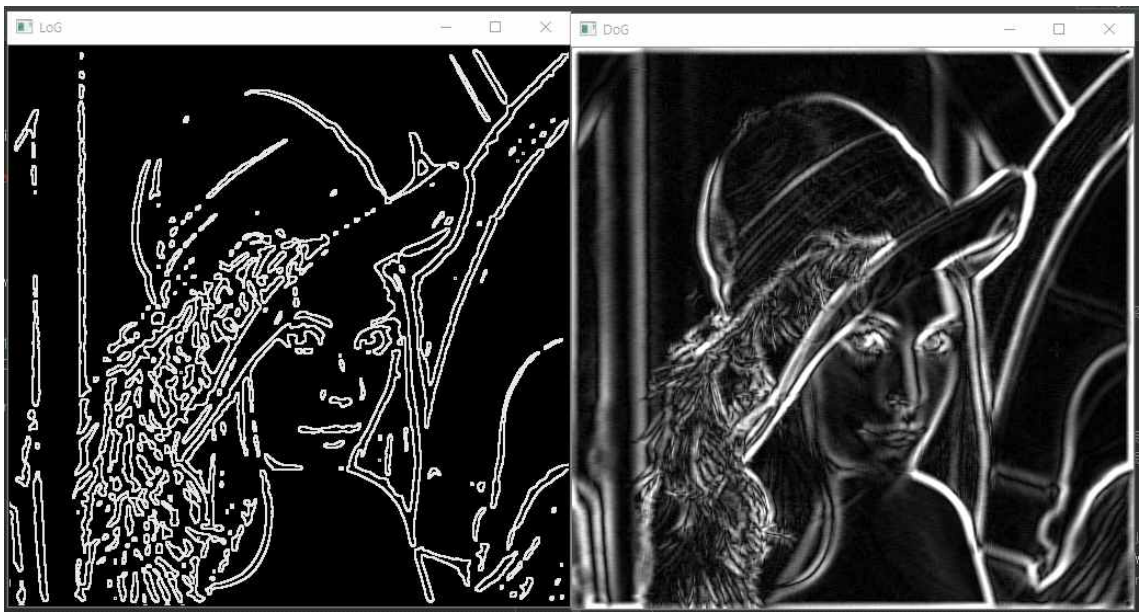
```python
def my_DoG(img, ksize, sigma = 1, gx = 0, boundary = 0): #default (3,3) sigma = 1, y축 편미분
    '''
    :param img: DoG edge detection을 수행할 이미지.
    :param ksize: Kernel size. ksize x 1 kernel 또는 1 x ksize kernel사용.
    :param sigma: Gaussian 분포에서 사용하는 표준편차.
    :param gx: 0 : y축 편미분, 1 : x축 편미분
    :param boundary: filtering 경계 처리 방법. 0 : zero-padding,(default) 1 : repetition, 2 : mirroring
    :return: 축에대한 미분 결과값 ( Gradient 값 )
    '''
    gesu = (1 / (2 * np.pi * np.square(sigma)))
    if (gx == 0):
        DoG = np.ones((1,ksize))
        middleX = ksize // 2
        for i in range(1):
            for j in range(ksize):
                DoG[i][j] = -1 * gesu * ((j-middleX)/(sigma ** 2)) * np.exp(-1 * ((np.square(j-middleX) / (2 * (sigma ** 2)))))

    elif (gx == 1):
        DoG = np.ones((ksize,1))
        middleY = ksize // 2
        for i in range(ksize):
            for j in range(1):
                DoG[i][j] = -1 * gesu * ((i-middleY)/(sigma ** 2)) * np.exp(-1 * ((np.square(i-middleY) / (2 * (sigma ** 2)))))

    DoG_img = my_filtering(img, DoG, boundary = boundary) # DoG 는 만들어진 kernel

    return DoG_img
```

DoG의 경우에도 식을 그대로 따라했다. y축 편미분의 경우 배열 크기가 (1,ksize)가 되도록 했고 x축 편미분일 경우 (ksize,1) 사이즈의 커널을 가진다.



왼쪽이 LoG 결과고 오른쪽이 DoG 결과다. sigma 값은 각각 2.5, 4.2로 설정했다.

느낀 점

DoG 부분이 조금 어려웠다.

과제 난이도
어렵다.