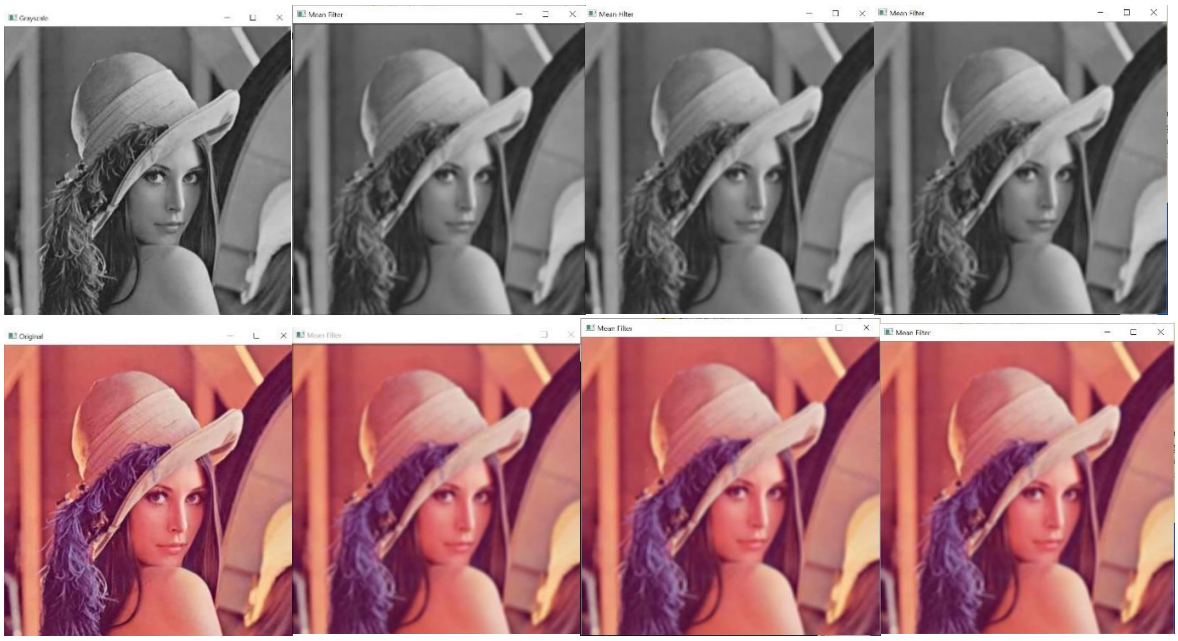
Technical Report

1829008 김민영

1. practice : Uniform mean filter (MeanFilterGray.cpp, MeanFilterRGB.cpp)

Meanfilter를 zero-paddle, mirroringm adjust kernel 방식으로 구현하고 이를 출력한다.



Kernel Size : n -> (2n+1)x(2n+1)

Kernel Matrix : uniform mean filter에서는 모든 kernel value가 같으므로 다음과 같이 초기화한다.

```
kernel = Mat::ones(kernel_size, kernel_size, CV_32F) / (float)(kernel_size *
kernel_size);
float kernelvalue = kernel.at<float>(0, 0);
```

Boundary processing : zero-padding, mirroring, adjustng the fiter kernel 세가지 방식으로 상황에 맞게 코드를 구현한다.

```
if (!strcmp(opt, "zero-paddle")) {
          float sum1 = 0.0;
                    for (int a = -n; a <= n; a++) { // for each kernel window
                              for (int b = -n; b <= n; b++) {
                                        if ((i + a <= row - 1) && (i + a >= 0) && (j + b <=
col - 1) && (j + b >= 0)) { //if the pixel is not a border pixel
                                                  sum1 += kernelvalue *
(float)(input.at<G>(i + a, j + b));
                                        }
                              }
                    }
          output.at<G>(i, j) = (G)sum1;
}
```

*Mirroring, adjust kernel도 skeleton 코드를 참조하여 구현하면 된다.

(MeanFilterGray.cpp,MeanFilterRGB 코드 참조)

Mean filter가 zero-paddle, mirroringm adjust kernel 모두에서 잘 적용된 것을 확인할 수 있다.

2. practice : gaussian filter (GuassianGray.cpp, GuassianRGB.cpp)

Gaussian filter를 구현하고 이를 출력한다.

Kernel size : Kernel Size : n -> (2n+1)x(2n+1)

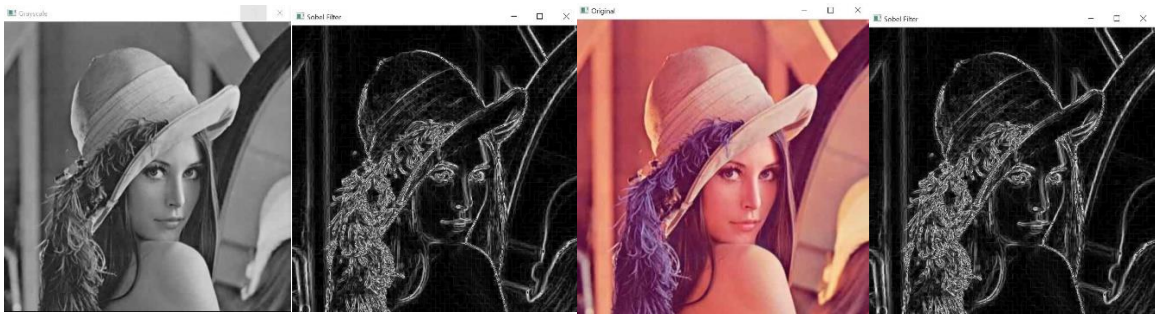
Kernel Matrix : gaussian filter에서는 정규분포 공식에 따라 값을 대입한다.

```
Mat output = Mat::zeros(row, col, input.type());
denom = 0.0;
for (int a = -n; a <= n; a++) {  // Denominator in m(s,t)
          for (int b = -n; b <= n; b++) {
                    float value1 = exp(-(pow(a, 2) / (2 * pow(sigmaS, 2))) - (pow(b, 2) /
(2 * pow(sigmaT, 2))));
                    kernel.at<float>(a + n, b + n) = value1;
                    denom += value1;
          }
}
for (int a = -n; a <= n; a++) {  // Denominator in m(s,t)
          for (int b = -n; b <= n; b++) {
                    kernel.at<float>(a + n, b + n) /= denom;
          }
}
```

이후 kernel value 를 불러와 practice1 과 비슷한 방식으로 Zero padding, mirroring, adjustkernel 방식에 맞게 구현하면 된다.(코드 참조)

Gaussian filter 가 mean filter 보다 좀더 자연스러운 방식이다.

3. practice : sobel filter (SobelGray.cpp, SobelRGB.cpp)



Kernel Size : n=1 (2*1+1)x(2*1+1)

Kernel Matrix : sobel filter에서는 다음과 같은 값을 대입한다.

```
Sx = Mat::zeros(3, 3, CV_32F);
Sy = Mat::zeros(3, 3, CV_32F);
Sx.at<float>(0, 0) = -1;
Sx.at<float>(0, 2) = 1;
Sx.at<float>(1, 0) = -2;
Sx.at<float>(1, 2) = 2;
Sx.at<float>(2, 0) = -1;
Sx.at<float>(2, 2) = 1;
Sy.at<float>(0, 0) = -1;
Sy.at<float>(0, 1) = -2;
Sy.at<float>(0, 2) = -1;
Sy.at<float>(2, 0) = 1;
Sy.at<float>(2, 1) = 2;
Sy.at<float>(2, 2) = 1;
```

S_x

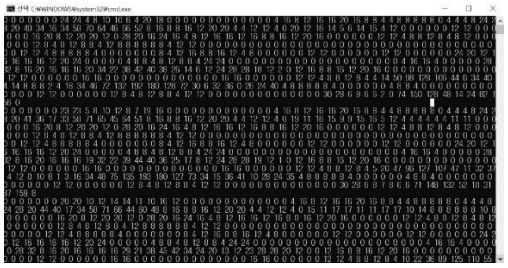
−1	0	1
−2	0	2
−1	0	1

S_y

−1	−2	−1
0	0	0
1	2	1

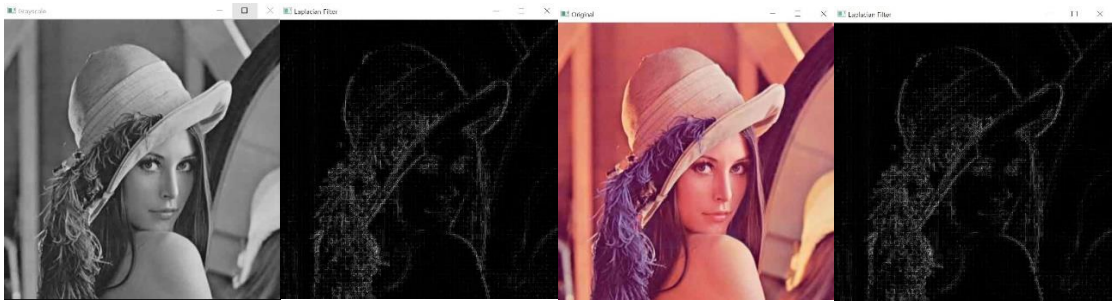
```
…생략
sum1 += sx * ((float)(input.at<G>(tempa, tempb)));
sum2 += sy * ((float)(input.at<G>(tempa, tempb)));
…생략
output.at<G>(i, j) = sqrt(sum1*sum1 + sum2 * sum2);
```

Sobel filter $M(x,y) = \sqrt{{I_x}^2 + {I_y}^2}$



-> output 값들은 normalize 를 통해 0~255 범위 사이로 조정해주었다.

4. practice : laplacian filter (LaplacianGray.cpp, LaplacianRGB.cpp)



Kernel Size : n=1 (2*1+1)x(2*1+1)

Kernel Matrix : laplacian filter에서는 다음과 같은 값을 대입한다.

Laplacian filter

0	1	0
1	−4	1
0	1	0

```
kernel = Mat::zeros(3, 3, CV_32F);
kernel.at<float>(0, 1) = 1;
kernel.at<float>(2, 1) = 1;
kernel.at<float>(1, 0) = 1;
kernel.at<float>(1, 2) = 1;
kernel.at<float>(1, 1) = -4;
```

```
...(생략)
      sum1 += kernelvalue * ((float)(input.at<G>(tempa, tempb)));
       }
}
sum1 = abs(sum1);
if (sum1 < 0) sum1 = 0;
if (sum1 > 255) sum1 = 255;
output.at<G>(i, j) = sum1;
...(생략)
```

$$O = |L * I|$$

Laplacian filter에서도 역시 normalize를 통해 값의 범위를 0~255로 조정해주었다.

5. practice : gaussian filter in a separable manner (GuassianGray_separable.cpp, GuassianRGB_separable.cpp)

기존 방식)

$$O(i,j) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) I(i+s, j+t) \qquad w(s,t) = \frac{1}{\sum_{m=-a}^{a} \sum_{n=-b}^{b} \exp\left(-\frac{m^2}{2{\sigma_s}^2} - \frac{n^2}{2{\sigma_t}^2}\right)} \exp\left(-\frac{s^2}{2{\sigma_s}^2} - \frac{t^2}{2{\sigma_t}^2}\right)$$

Separable 방식)

$$O(i,j) = \sum_{s=-a}^{a} w_s(s) \sum_{t=-b}^{b} w_t(t)\, I(i+s, j+t)$$

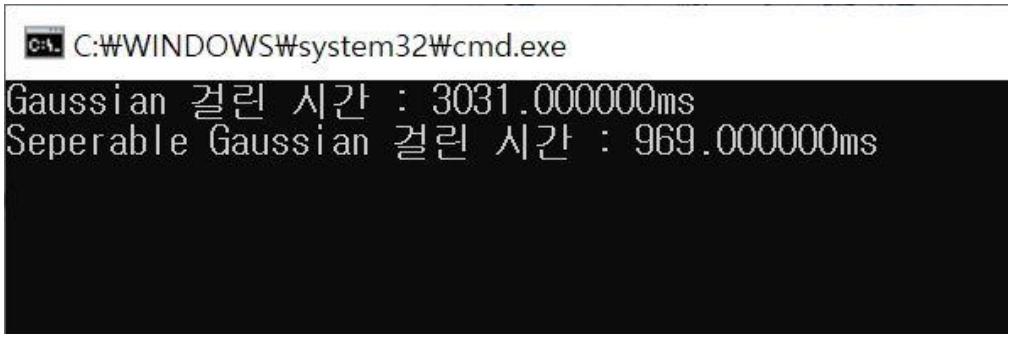
$$w_s(s) = \frac{1}{\sum_{m=-a}^{a} \exp\left(-\frac{m^2}{2{\sigma_s}^2}\right)} \exp\left(-\frac{s^2}{2{\sigma_s}^2}\right)$$

$$w_t(t) = \frac{1}{\sum_{n=-b}^{b} \exp\left(-\frac{n^2}{2{\sigma_t}^2}\right)} \exp\left(-\frac{t^2}{2{\sigma_t}^2}\right)$$

```
for (int a = -n; a <= n; a++) {
        float value_s = exp(-(pow(a, 2) / (2 * pow(sigmaS, 2))));
        kernel_s.at<float>(a + n,0) = value_s;
        denom_s += value_s;
}
for (int b = -n; b <= n; b++) {
        float value2 = exp(-(pow(b, 2) / (2 * pow(sigmaT, 2))));
        kernel_t.at<float>(0,b+n) = value2;
        denom_t += value2;
}
for (int a = -n; a <= n; a++) {
        kernel_s.at<float>(a + n, 0) /= denom_s;
}
for (int b = -n; b <= n; b++) {
        kernel_t.at<float>(0, b + n) /= denom_t;
}
```

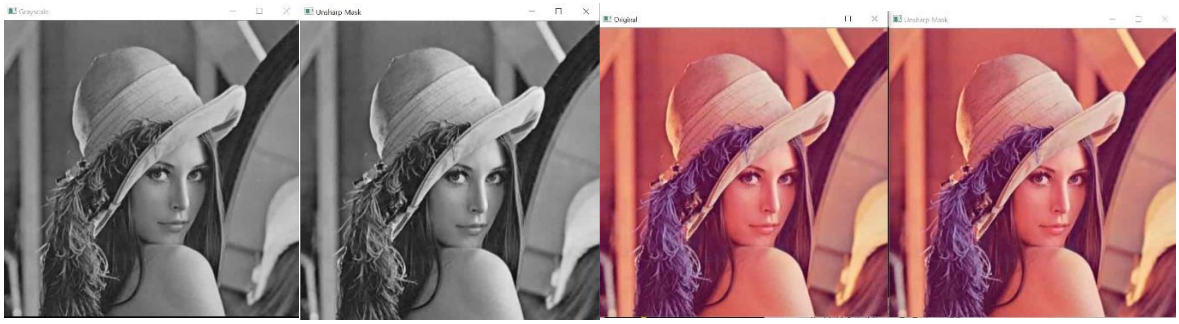
Convolution -> 코드 참조

위와 같이 Separable한 성질을 이용하여 기존의 gaussian filter를 수정하여 fast_gaussianfilter 함수를 만들었다. 또한 시간을 재본 결과, Separable 방식이 gaussian filter보다 빨랐음을 확인할 수 있다.

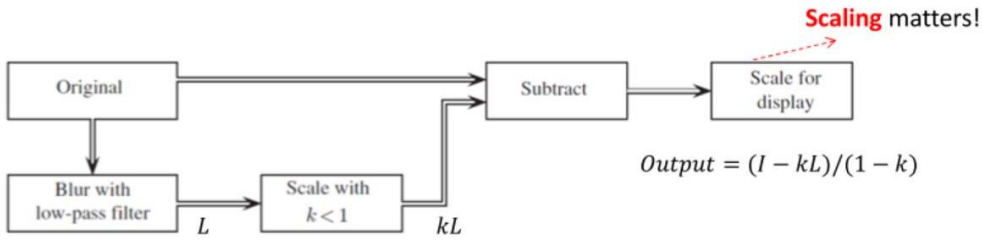


6. homework : unsharp masking (UnsharpMasking_Gray.cpp, UnsharpMasking_RGB.cpp)

Gaussian filter를 low-pass filter로 사용하여 unsharp Masking을 구현한다.



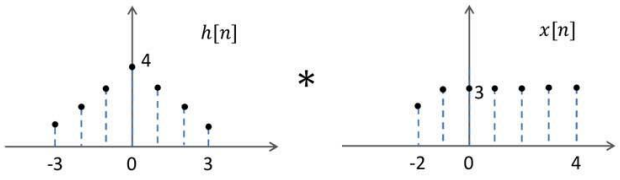
코드 설명




```
Mat UnsharpMask(const Mat input, int n, float sigmaT, float sigmaS, const char* opt, int
k) {
	int row = input.rows;
	int col = input.cols;
	Mat L = gaussianfilter(input, 1, 1, 1, "zero-paddle");
	Mat output = Mat::zeros(row, col, input.type());
	for (int i = 0; i < row; i++) {
		for (int j = 0; j < col; j++) {
			// output = (I-kL)/(1-k)
			output.at<G>(i, j)=(input.at<G>(i, j)-k*L.at<G>(i, j))/(1-k);
		}
	}
	return output;
}
```

➔ Unsharp masking 을 통해 사진이 좀더 sharp 해지는 것을 확인할 수 있다.

7. homework : convolution 계산



$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m]$$

$y[-5] = x[-5] * h[-5] = \sum_{m=-\infty}^{\infty} x[m]h[-5-m]$

$= x[-2]h[-3]$

$= 2 \times 1 = 2$

$y[-4] = x[-4] * h[-4] = \sum_{m=-\infty}^{\infty} x[m]h[-4-m]$

$= x[-1]h[-3] + x[-2]h[-2]$

$= 3 \times 1 + 2 \times 2 = 7$

$y[-3] = x[-3] * h[-3] = \sum_{m=-\infty}^{\infty} x[m]h[-3-m]$

$= x[0]h[-3] + x[-1]h[-2] + x[-2]h[-1]$

$= 3(1+2) + 2 \times 3 = 15$

$y[-2] = x[-2] * h[-2] = \sum_{m=-\infty}^{\infty} x[m]h[-2-m]$

$= x[1]h[-3] + x[0]h[-2] + x[-1]h[-1] + x[-2]h[0]$

$= 3(1+2+3) + 2 \times 4 = 18+8 = 26$

$y[-1] = x[-1] * h[-1] = \sum_{m=-\infty}^{\infty} x[m]h[-1-m]$

$= x[2]h[-3] + x[1]h[-2] + x[0]h[-1] + x[-1]h[0] + x[-2]h[-1]$

$= 3(1+2+3+4) + 2 \times 3 = 36$

$y[0] = x[0] * h[0] = \sum_{m=-\infty}^{\infty} x[m]h[-m]$

$= x[-2]h[2] + x[-1]h[1] + x[0]h[0] + x[1]h[-1] + x[2]h[-2] + x[3]h[-3]$

$= 2 \times 2 + 3 \times 3 + 3 \times 4 + 3 \times 3 + 3 \times 2 + 3 \times 1$

$= 43$

$y[1] = x[1] * h[1] = \sum_{m=-\infty}^{\infty} x[m]h[1-m]$

$= x[-2]h[3] + x[-1]h[2] + x[0]h[1] + x[1]h[0] + x[2]h[-1] + x[3]h[-2] + x[4]h[-3]$

$= 2 \times 1 + 3(2+3+4+3+2+1)$

$= 2 + 3 \times 15 = 47$

$y[2] = x[2] * h[2] = \sum_{m=-\infty}^{\infty} x[m]h[2-m]$

$= x[-1]h[3] + x[0]h[2] + x[1]h[1] + x[2]h[0] + x[3]h[-1] + x[4]h[-2]$

$= 3(1+2+3+4+3+2) = 45$

$y[3] = x[3] * h[3] = \sum_{m=-\infty}^{\infty} x[m]h[3-m]$

$= x[0]h[3] + x[1]h[2] + x[2]h[1] + x[3]h[0] + x[4]h[-1]$

$= 3(1+2+3+4+3) = 3(13) = 39$

$y[4] = x[4] * h[4] = \sum_{m=-\infty}^{\infty} x[m]h[4-m]$

$= x[1]h[3] + x[2]h[2] + x[3]h[1] + x[4]h[0]$

$= 3(1+2+3+4) = 30$

$y[5] = x[5] * h[5] = \sum_{m=-\infty}^{\infty} x[m]h[5-m]$

$= x[2]h[3] + x[3]h[2] + x[4]h[1]$

$= 3(1+2+3) = 18$

$y[6] = x[6] * h[6] = \sum_{m=-\infty}^{\infty} x[m]h[6-m]$

$= x[3]h[3] + x[4]h[2]$

$= 3(1+2) = 9$

$y[7] = x[7] * h[7] = \sum_{m=-\infty}^{\infty} x[m]h[7-m]$

$= x[4]h[3]$

$= 3 \times 1 = 3$

$y[-5] = 2$
$y[-4] = 7$
$y[-3] = 15$
$y[-2] = 26$
$y[-1] = 36$
$y[0] = 43$
$y[1] = 47$
$y[2] = 45$
$y[3] = 39$
$y[4] = 30$
$y[5] = 18$
$y[6] = 9$
$y[7] = 3$