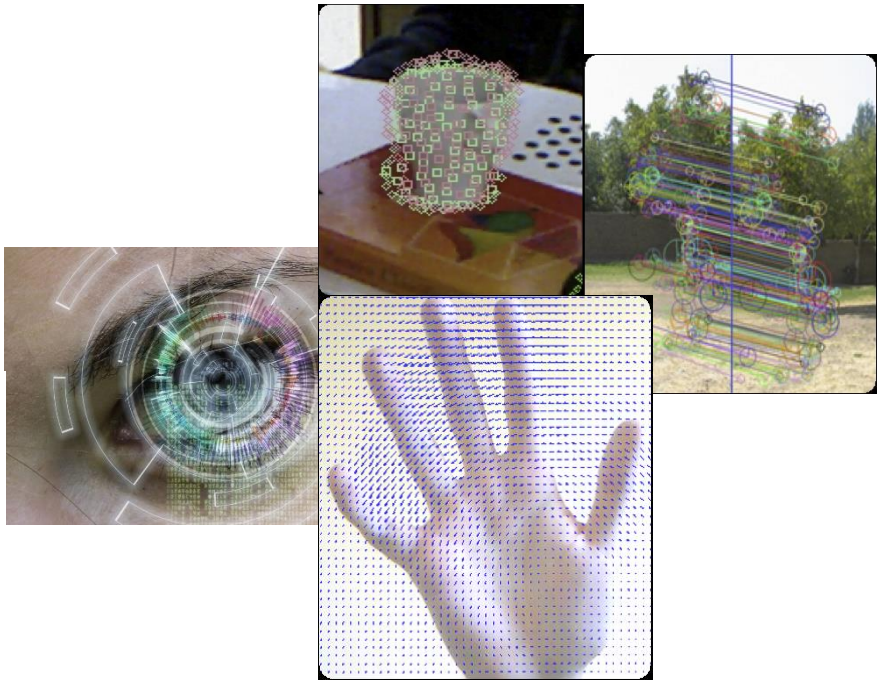
IVPL
Intelligent Vision Processing Lab

2023 Fall

COMPUTER VISION 비젼 프로그래밍



5장. 영상필터링(image filtering)(Practice)

Convolution processing (컨벌루션 처리) 기반 필터링

- Filtering(필터링)을 위한 다양한 API가 존재함
  - Blurring(블러링)
  - Sharpening (샤프닝)
  - 에지 검출 필터
  - 라플라시안 필터
  - 직접 디자인 가능한 filter2D 함수 등

- Blurring 필터
  - void **blur**(InputArray **src**, OutputArray **dst**, Size **ksize**, Point **anchor**=Point(-1,-1), int **borderType**=BORDER_DEFAULT)

Parameters:	•**src** – input image; it can have any number of channels, which are processed in dependently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F. •**dst** – output image of the same size and type as src. •**ksize** – blurring kernel size. •**anchor** – anchor point; default value Point(-1,-1) means that the anchor is at the kernel center. •**borderType** – border mode used to extrapolate pixels outside of the image.

Convolution processing (컨벌루션 처리) 기반 필터링

- void **boxFilter**(InputArray **src**, OutputArray **dst**, int **ddepth**, Size **ksize**, Point **anchor**=Point(-1,-1), bool **normalize**=true, int **borderType**=BORDER_DEFAULT)

Parameters:	•**src** – input image. •**dst** – output image of the same size and type as src. •**ddepth** – the output image depth (-1 to use src.depth()). •**ksize** – blurring kernel size. •**anchor** – anchor point; default value Point(-1,-1) means that the anchor is at t he kernel center. •**normalize** – flag, specifying whether the kernel is normalized by its area or not . •**borderType** – border mode used to extrapolate pixels outside of the image.

- void **GaussianBlur**(InputArray **src**, OutputArray **dst**, Size **ksize**, double **sigmaX**, double **sigmaY**=0, int **borderType**=BORDER_DEFAULT)

Parameters:	•**src** – input image; the image can have any number of channels, which are processed indepe ndently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F. •**dst** – output image of the same size and type as src. •**ksize** – Gaussian kernel size. ksize.width and ksize.height can differ but they both must be p ositive and odd. Or, they can be zero's and then they are computed from sigma* . •**sigmaX** – Gaussian kernel standard deviation in X direction. •**sigmaY** – Gaussian kernel standard deviation in Y direction; if sigmaY is zero, it is set to be equal to sigmaX, if both sigmas are zeros, they are computed from ksize.width and ksize.hei ght , respectively (see getGaussianKernel() for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of ksize, s igmaX, and sigmaY. •**borderType** – pixel extrapolation method (see borderInterpolate for details).

Convolution processing (컨벌루션 처리) 기반 필터링

- void **medianBlur**(InputArray **src**, OutputArray **dst**, int **ksize**)

Parameters:	•**src** – input 1-, 3-, or 4-channel image; when ksize is 3 or 5, the image depth should be CV_8U, CV_16U, or CV_32F, for larger aperture sizes, it can only be CV_8U. •**dst** – destination array of the same size and type as src. •**ksize** – aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7 ...

- **Smooth 함수 (opencv 1.x~)**

Convolution processing (컨벌루션 처리) 기반 필터링

filter2D 함수 ← 사용자가 kernel 설계 가능

- void **filter2D**(InputArray **src**, OutputArray **dst**, int **ddepth**, InputArray **kernel**, Point **anchor**=Point(-1,-1), double **delta**=0, int **borderType**=BORDER_DEFAULT)

Parameters:

•**src** – input image.
•**dst** – output image of the same size and the same number of channels as src.
•**ddepth** – desired depth of the destination image; if it is negative, it will be the same as src.depth(); the following combinations of src.depth() and ddepth are s upported:

- src.depth() = CV_8U, ddepth = -1/CV_16S/CV_32F/CV_64F
- src.depth() = CV_16U/CV_16S, ddepth = -1/CV_32F/CV_64F
- src.depth() = CV_32F, ddepth = -1/CV_32F/CV_64F
- src.depth() = CV_64F, ddepth = -1/CV_64F

•when ddepth=-1, the output image will have the same depth as the source.
•**kernel** – convolution kernel (or rather a correlation kernel), a single-channel flo ating point matrix; if you want to apply different kernels to different channels, s plit the image into separate color planes using split() and process them individ ually.
•**anchor** – anchor of the kernel that indicates the relative position of a filtered p oint within the kernel; the anchor should lie within the kernel; default value (-1, -1) means that the anchor is at the kernel center.
•**delta** – optional value added to the filtered pixels before storing them in dst.
•**borderType** – pixel extrapolation method (see borderInterpolate for details).

Convolution processing (컨벌루션 처리) 기반 필터링

- Blurring 관련 기본 예제

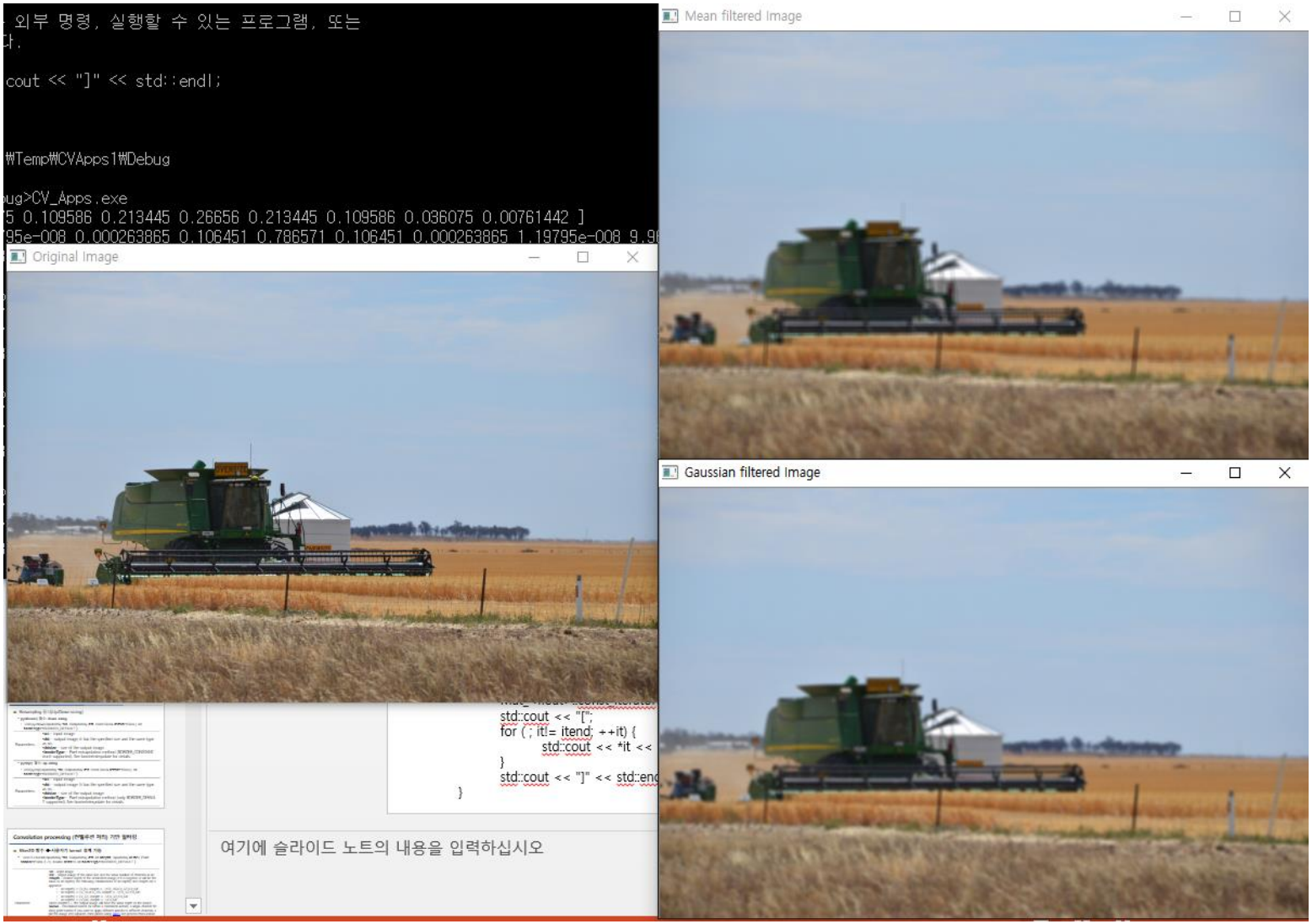
```
Int main(~){
        (계속)
        // Display the image
        :namedWindow("Original Image");
        :imshow("Original Image",image);

        // Blur the image with a mean filter
        blur(image,result,cv::Size(5,5));

        // Display the blurred image
        namedWindow("Mean filtered Image");
        imshow("Mean filtered Image",result);

        // Gaussian Blur the image
        GaussianBlur(image,result,cv::Size(5,5),1.5);

        // Display the blurred image
        namedWindow("Gaussian filtered Image");
        imshow("Gaussian filtered Image",result);

        // Get the gaussian kernel (1.5)
        Mat gauss= cv::getGaussianKernel(9,1.5,CV_32F);

        // Display kernel values
        Mat_<float>::const_iterator it= gauss.begin<float>();
        Mat_<float>::const_iterator itend= gauss.end<float>();
        std::cout << "[";
        for ( ; it!= itend; ++it) {
                std::cout << *it << " ";
        }
        std::cout << "]" << std::endl;
}
```

Mat **getGaussianKernel**(int **ksize**, double **sigma**,
int **ktype**=CV_64F)

<Parameters:>
ksize – Aperture size. It should be odd and positive.
sigma – Gaussian standard deviation. If it is non-positive, it is computed from ksize as sigma = 0.3*((ksize-1)*0.5 - 1) + 0.8 .
ktype – Type of filter coefficients. It can be CV_32f or CV_64F .

Convolution processing (컨벌루션 처리) 기반 필터링

- 수행 결과:



Convolution processing (컨벌루션 처리) 기반 필터링

- Noise 제거 실습: medianBlur와의 비교

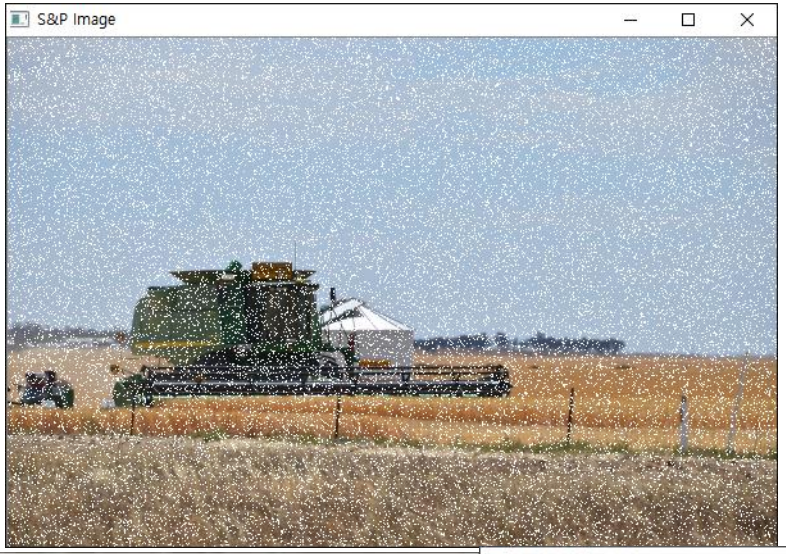
```
int main( int argc, char** argv )
{
        Mat image, result, dst, dst1;

        /// Load image
        image = imread( "test.jpg", 1);   // Read the file

         if (image.empty()){     // Check for invalid input
         cout << "Could not open or find the image"
                                          << std::endl;
         return -1;
         }

        // make a noise image
      salt(image, 30000);

      // Display the S&P image
      cv::namedWindow("S&P Image");
      cv::imshow("S&P Image",image);

      // Blur the image with a mean filter
      cv::blur(image,result,cv::Size(5,5));

      // Display the blurred image
      cv::namedWindow("Mean filtered S&P Image");
      cv::imshow("Mean filtered S&P Image",result);
```

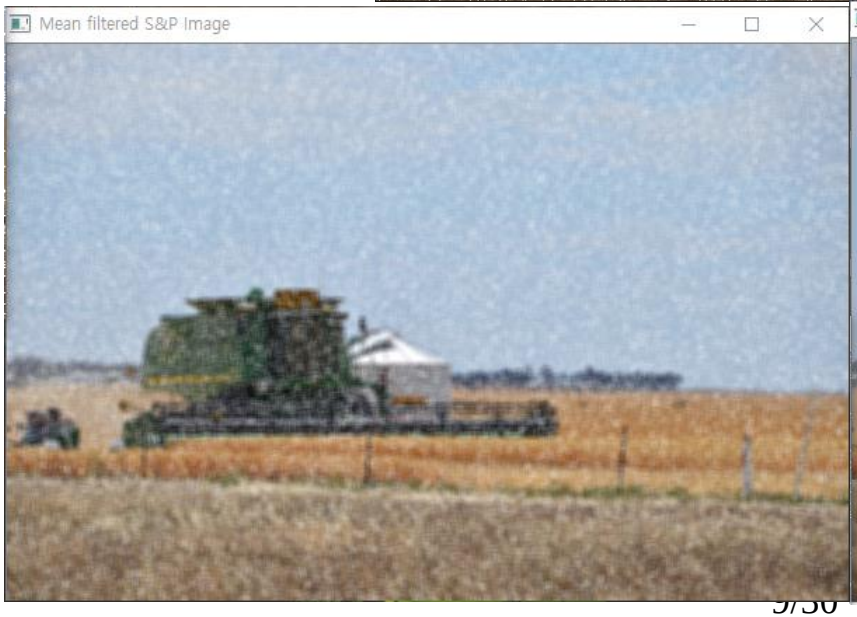
(계속)

```
      // Applying a median filter
      cv::medianBlur(image,result,5);

      // Display the blurred image
      cv::namedWindow("Median filtered S&P Image");
      cv::imshow("Median filtered S&P Image",result);

        waitKey(0);

        return 0;
}
```

Convolution processing (컨벌루션 처리) 기반 필터링

- 수행 결과:

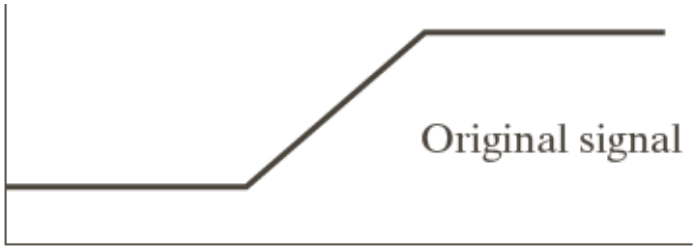




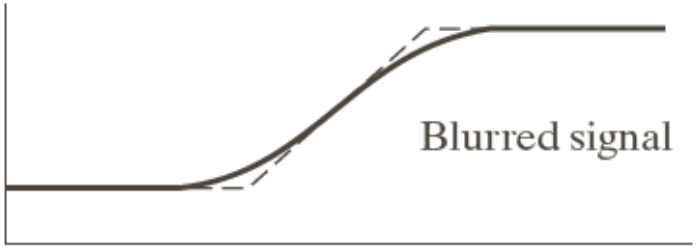
Convolution processing (컨벌루션 처리) 기반 필터링

- 일반적인 Sharpening 과정
  - Unsharp Mask 이용

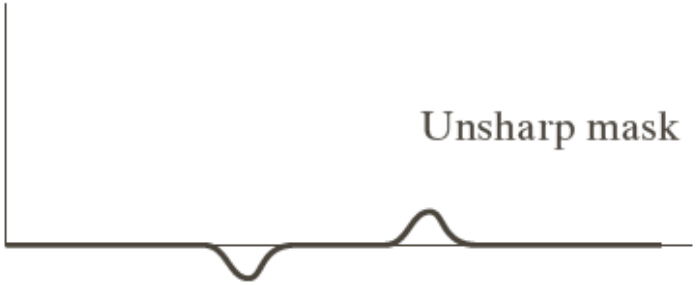
$f(x,y)$



$\overline{f}(x,y)$



$$g_{mask}(x,y) = f(x,y) - \overline{f}(x,y)$$



$$g(x,y) = f(x,y) + k * g_{mask}(x,y)$$

k>1: highboost filtering

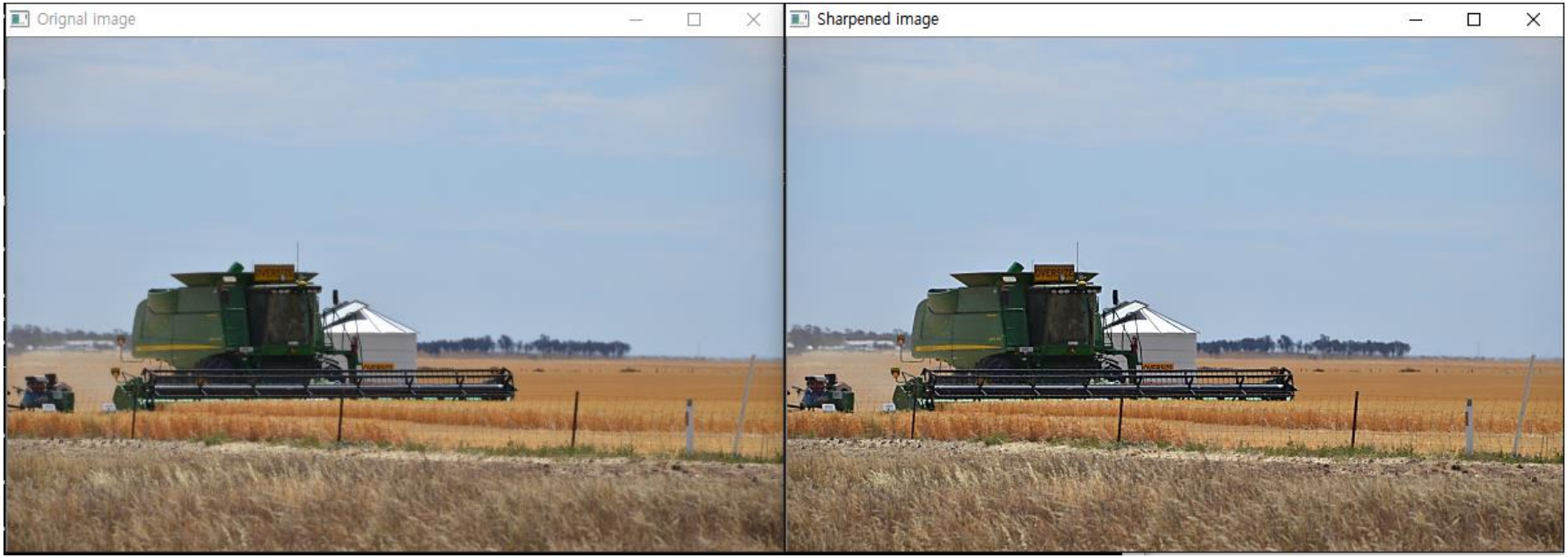
Convolution processing (컨벌루션 처리) 기반 필터링

- Unsharp Mask 이용 실습

```
int main( int argc, char** argv )
{
        Mat image, result, dst, dst1;

        /// Load image
        image = imread( "test.jpg", 1);    // Read the file

         if (image.empty()){                              // Check for invalid input
                cout << "Could not open or find the image" << std::endl;
                return -1;
         }

        // Display the S&P image
      cv::namedWindow("Orignal image");
      cv::imshow("Orignal image",image);

      // sharpen image using "unsharp mask" algorithm
      Mat blurred; double sigma = 1, threshold = 5, amount = 1;
      GaussianBlur(image, blurred, Size(), sigma, sigma);
      Mat lowContrastMask = abs(image - blurred) < threshold;
      Mat sharpened = image*(1+amount) + blurred*(-amount);
      image.copyTo(sharpened, lowContrastMask);

      cv::namedWindow("Sharpened image");
      cv::imshow("Sharpened image",sharpened);

        waitKey(0);

        return 0;
}
```

Convolution processing (컨벌루션 처리) 기반 필터링

- Unsharp Mask 이용 결과



Convolution processing (컨벌루션 처리) 기반 필터링

- Resampling 필터(Up/Down sizing)
  - **pyrDown() 함수: down sizing**
    - void **pyrDown**(InputArray **src**, OutputArray **dst**, const Size& **dstsize**=Size(), int **borderType**=BORDER_DEFAULT)

Parameters:	•**src** – input image. •**dst** – output image; it has the specified size and the same type as src. •**dstsize** – size of the output image. •**borderType** – Pixel extrapolation method (BORDER_CONSTANT don't supported). See borderInterpolate for details.

  - **pyrUp() 함수: up sizing**
    - void **pyrUp**(InputArray **src**, OutputArray **dst**, const Size& **dstsize**=Size(), int **borderType**=BORDER_DEFAULT)

Parameters:	•**src** – input image. •**dst** – output image. It has the specified size and the same type as src . •**dstsize** – size of the output image. •**borderType** – Pixel extrapolation method (only BORDER_DEFAULT supported). See borderInterpolate for details.

Convolution processing (컨벌루션 처리) 기반 필터링

filter2D 함수 ← 사용자가 kernel 설계 가능

- void **filter2D**(InputArray **src**, OutputArray **dst**, int **ddepth**, InputArray **kernel**, Point **anchor**=Point(-1,-1), double **delta**=0, int **borderType**=BORDER_DEFAULT)

Parameters:

•**src** – input image.
•**dst** – output image of the same size and the same number of channels as src.
•**ddepth** – desired depth of the destination image; if it is negative, it will be the same as src.depth(); the following combinations of src.depth() and ddepth are s upported:

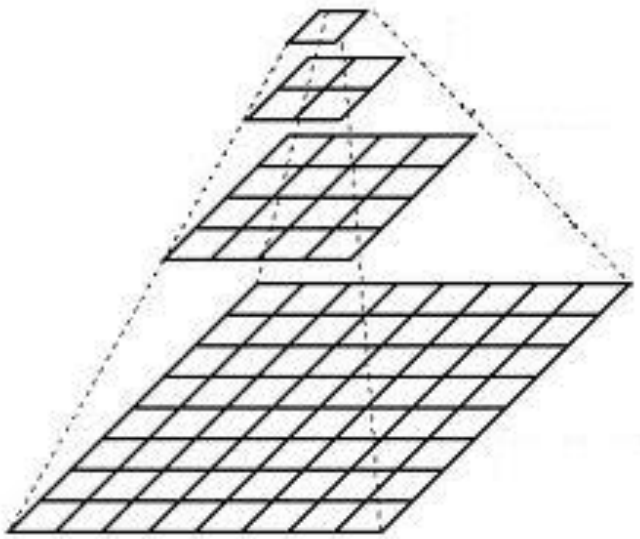
- src.depth() = CV_8U, ddepth = -1/CV_16S/CV_32F/CV_64F
- src.depth() = CV_16U/CV_16S, ddepth = -1/CV_32F/CV_64F
- src.depth() = CV_32F, ddepth = -1/CV_32F/CV_64F
- src.depth() = CV_64F, ddepth = -1/CV_64F

•when ddepth=-1, the output image will have the same depth as the source.
•**kernel** – convolution kernel (or rather a correlation kernel), a single-channel flo ating point matrix; if you want to apply different kernels to different channels, s plit the image into separate color planes using split() and process them individ ually.
•**anchor** – anchor of the kernel that indicates the relative position of a filtered p oint within the kernel; the anchor should lie within the kernel; default value (-1, -1) means that the anchor is at the kernel center.
•**delta** – optional value added to the filtered pixels before storing them in dst.
•**borderType** – pixel extrapolation method (see borderInterpolate for details).

Convolution processing (컨벌루션 처리) 기반 필터링

- Resampling 실습

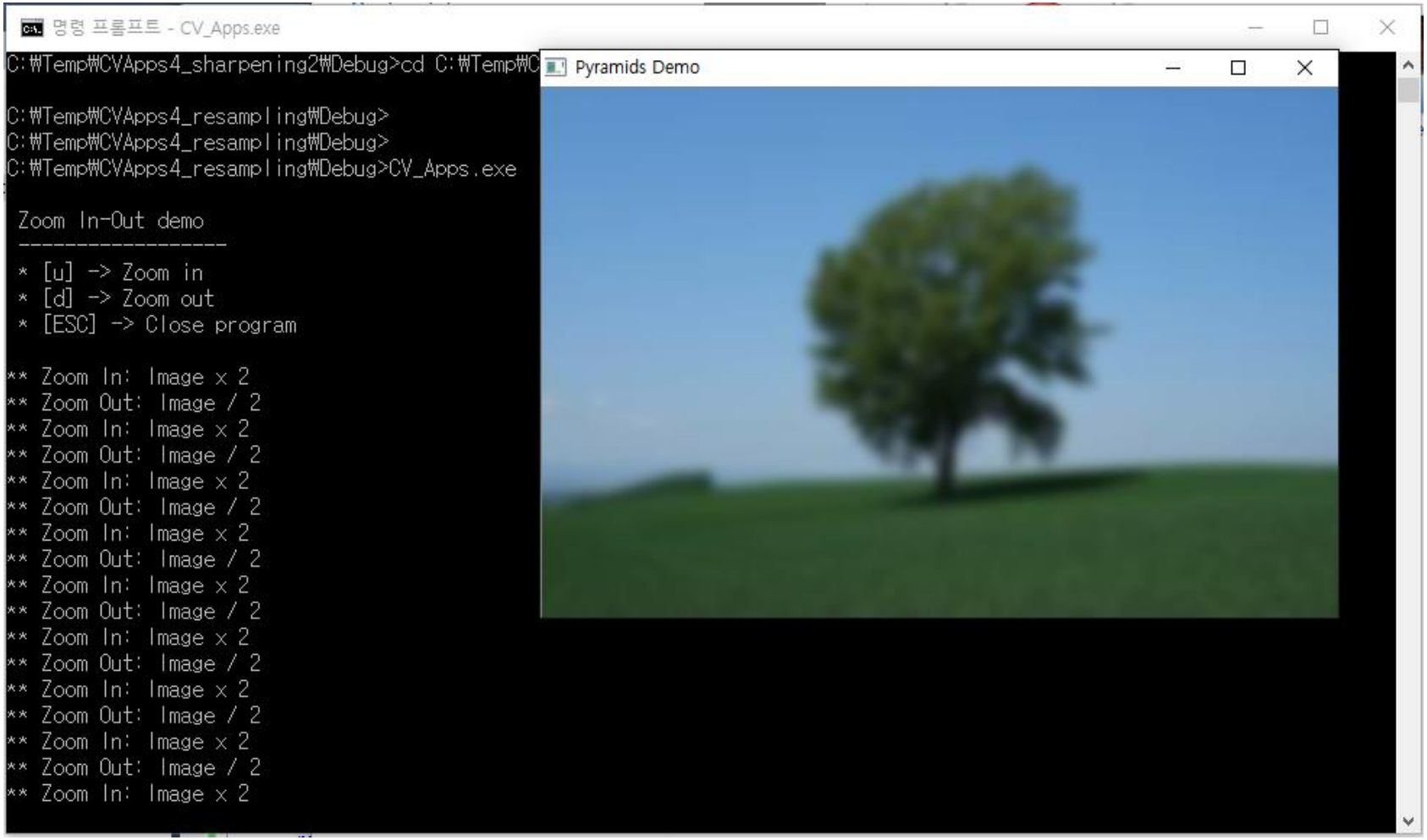
```
int main( int argc, char** argv )
{

        Mat src, dst, tmp;
        char* window_name = "Pyramids Demo";

        /// General instructions
        printf( "₩n Zoom In-Out demo  ₩n " );
        printf( "------------------ ₩n" );
        printf( " * [u] -> Zoom in  ₩n" );
        printf( " * [d] -> Zoom out ₩n" );
        printf( " * [ESC] -> Close program ₩n ₩n" );

        /// Test image - Make sure it s divisible by 2^{n}
        src = imread( "test2.jpg" );
        if( !src.data )
        {      printf(" No data! -- Exiting the program ₩n");
               return -1; }

        tmp = src;
        dst = tmp;
(계속)
```



Convolution processing (컨벌루션 처리) 기반 필터링

```
    /// Create window
    namedWindow( "Original Image", CV_WINDOW_AUTOSIZE );
    imshow( "Original Image", src );
    /// Loop
    while( true ){
        int c;
        c = waitKey(10);

        if( (char)c == 27 ) { break; } // ESC 키 입력 시 종료
        if( (char)c == 'u' )
          { pyrUp( tmp, dst, Size( tmp.cols*2, tmp.rows*2 ) );
            printf( "** Zoom In: Image x 2 ₩n" );
          }else if( (char)c == 'd' ) {
           pyrDown( tmp, dst, Size( tmp.cols/2, tmp.rows/2 ) );
           printf( "** Zoom Out: Image / 2 ₩n" );
         }

        imshow( window_name, dst );
        tmp = dst;
    }
    return 0;
}
```

Convolution processing (컨벌루션 처리) 기반 필터링



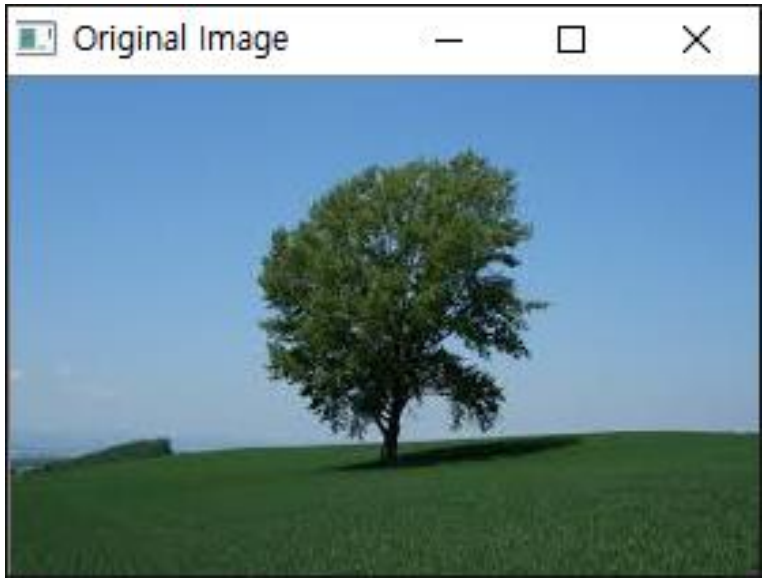
Convolution processing (컨벌루션 처리) 기반 필터링

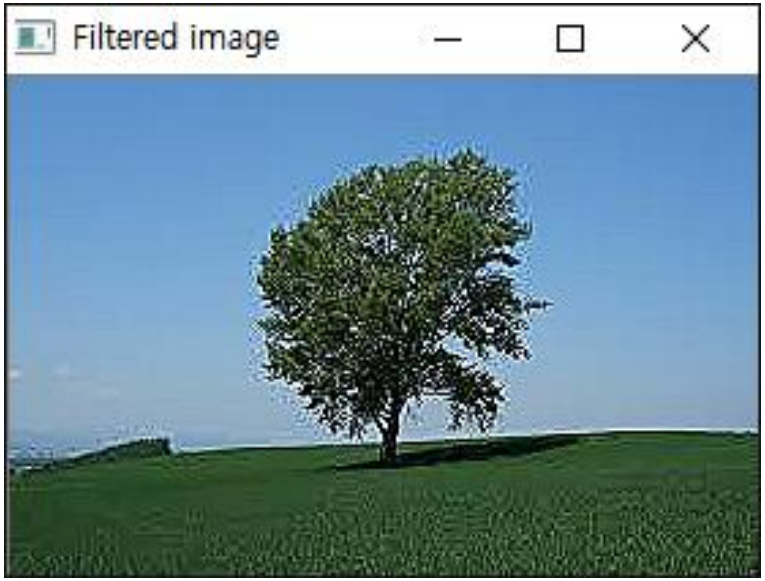
- filter2D 이용 실습

```
int main( int argc, char** argv )
{
        Mat image, result;
        image = imread( "test2.jpg" );

        // Construct kernel (all entries initialized to 0)
        cv::Mat kernel(3,3,CV_32F,cv::Scalar(0));

        // assigns kernel values
        kernel.at<float>(1,1)= 5.0;
        kernel.at<float>(0,1)= -1.0;
        kernel.at<float>(2,1)= -1.0;
        kernel.at<float>(1,0)= -1.0;
        kernel.at<float>(1,2)= -1.0;

        //filter the image
        filter2D(image,result,image.depth(),kernel);

        /// Create window
        namedWindow( "Original Image", CV_WINDOW_AUTOSIZE );
        imshow( "Original Image", image );

        /// Create window
        namedWindow( "Filtered image", CV_WINDOW_AUTOSIZE );
        imshow( "Filtered image", result );

        waitKey(0);

          return 0;
}
```

Convolution processing (컨벌루션 처리) 기반 필터링

- 수행 결과: 선명화 됨





Convolution processing (컨벌루션 처리) 기반 필터링

sepFilter2D 함수 ← linear kernel 2개를 분리하여 적용 가능

- void **sepFilter2D**(InputArray **src**, OutputArray **dst**, int **ddepth**, InputArray **kernelX**, InputArray **kernelY**, Point **anchor**=Point(-1,-1), double **delta**=0, int **borderType**=BORDER_DEFAULT)

Parameters:

•**src** – Source image.
•**dst** – Destination image of the same size and the same number of channels as src .
•**ddepth** – Destination image depth. The following combination of src.depth() and ddepth are supported:
- src.depth() = CV_8U, ddepth = -1/CV_16S/CV_32F/CV_64F
- src.depth() = CV_16U/CV_16S, ddepth = -1/CV_32F/CV_64F
- src.depth() = CV_32F, ddepth = -1/CV_32F/CV_64F
- src.depth() = CV_64F, ddepth = -1/CV_64F

•when ddepth=-1, the destination image will have the same depth as the source.
•**kernelX** – Coefficients for filtering each row.
•**kernelY** – Coefficients for filtering each column.
•**anchor** – Anchor position within the kernel. The default value means that the anchor is at the kernel center.
•**delta** – Value added to the filtered results before storing them.
•**borderType** – Pixel extrapolation method. See borderInterpolate for details.

Convolution processing (컨벌루션 처리) 기반 필터링: 에지

1차 미분 필터

Sobel 함수

- void **Sobel**(InputArray **src**, OutputArray **dst**, int **ddepth**, int **dx**, int **dy**, int **ksize**=3, double **scale**=1, double **delta**=0, int **borderType**=BORDER_DEFAULT)

Parameters:

•**src** – input image.
•**dst** – output image of the same size and the same number of channels as src .
•**ddepth** – output image depth; the following combinations of src.depth() and d depth are supported:
- src.depth() = CV_8U, ddepth = -1/CV_16S/CV_32F/CV_64F
- src.depth() = CV_16U/CV_16S, ddepth = -1/CV_32F/CV_64F
- src.depth() = CV_32F, ddepth = -1/CV_32F/CV_64F
- src.depth() = CV_64F, ddepth = -1/CV_64F

•when ddepth=-1, the destination image will have the same depth as the sourc e; in the case of 8-bit input images it will result in truncated derivatives.
•**xorder** – order of the derivative x.
•**yorder** – order of the derivative y.
•**ksize** – size of the extended Sobel kernel; it must be 1, 3, 5, or 7.
•**scale** – optional scale factor for the computed derivative values; by default, no scaling is applied (see getDerivKernels() for details).
•**delta** – optional delta value that is added to the results prior to storing them i n dst.
•**borderType** – pixel extrapolation method (see borderInterpolate for details).

Convolution processing (컨벌루션 처리) 기반 필터링: 에지

- **Scharr 함수:** the first x- or y- image derivative using Scharr operator
  - void **Scharr**(InputArray **src**, OutputArray **dst**, int **ddepth**, int **dx**, int **dy**, double **scale**=1, double **delta**=0, int **borderType**=BORDER_DEFAULT)

Parameters:	•**src** – input image. •**dst** – output image of the same size and the same number of channels as src. •**ddepth** – output image depth (see Sobel() for the list of supported combination of src.depth() and ddepth). •**dx** – order of the derivative x. •**dy** – order of the derivative y. •**scale** – optional scale factor for the computed derivative values; by default, no scaling is applied (see getDerivKernels() for details). •**delta** – optional delta value that is added to the results prior to storing them in dst. •**borderType** – pixel extrapolation method (see borderInterpolate for details).

- `Scharr(src, dst, ddepth, dx, dy, scale, delta, borderType)` =

  `Sobel(src, dst, ddepth, dx, dy, CV_SCHARR, scale, delta, borderType).`

Convolution processing (컨벌루션 처리) 기반 필터링: 에지

- Sobel edge 기본 예제

```
int main( int argc, char** argv ){

    Mat image, result;
    image = imread( "test.jpg", 0 );
    if (!image.data)
        return 0;

// Display the image
cv::namedWindow("Original Image");
cv::imshow("Original Image",image);

// Compute Sobel X derivative
cv::Mat sobelX;
cv::Sobel(image,sobelX,CV_8U,1,0,3,0.4,128);

// Display the image
cv::namedWindow("Sobel X Image");
cv::imshow("Sobel X Image",sobelX);

// Compute Sobel Y derivative
cv::Mat sobelY;
cv::Sobel(image,sobelY,CV_8U,0,1,3,0.4,128);
```

(계속)

Convolution processing (컨벌루션 처리) 기반 필터링: 에지

```
        // Compute norm of Sobel
        cv::Sobel(image,sobelX,CV_16S,1,0);
        cv::Sobel(image,sobelY,CV_16S,0,1);
        cv::Mat sobel;
        //compute the L1 norm
        sobel= abs(sobelX)+abs(sobelY);

        double sobmin, sobmax;
        cv::minMaxLoc(sobel,&sobmin,&sobmax);

        cv::Mat sobelImage;
        sobel.convertTo(sobelImage,CV_8U,-255./sobmax,255);

        // Display the image
        cv::namedWindow("Sobel Image");
        cv::imshow("Sobel Image",sobelImage);

        // Apply threshold to Sobel norm (low threshold value)
        cv::Mat sobelThresholded;
        cv::threshold(sobelImage, sobelThresholded, 225, 255, cv::THRESH_BINARY);

        // Display the image
        cv::namedWindow("Binary Sobel Image (low)");
        cv::imshow("Binary Sobel Image (low)",sobelThresholded);

        waitKey(0);

          return 0;
}
```

void Mat::**convertTo**(OutputArray m, int rtype,
double alpha=1, double beta=0)

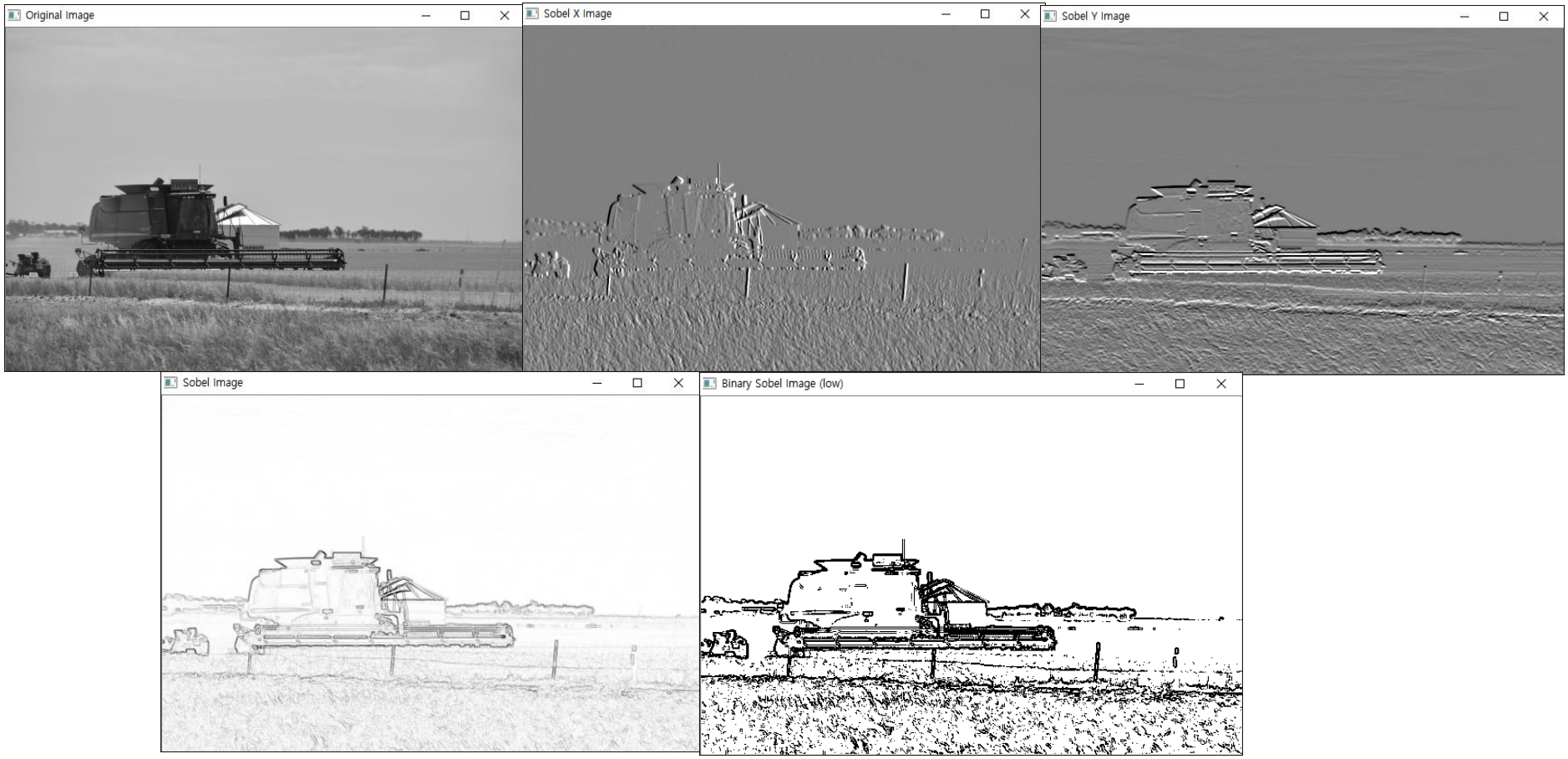
Parameters:

- **m** – output matrix; if it does not have a proper size or type before the operation, it is reallocated.
- **rtype** – desired output matrix type or, rather, the depth since the number of channels are the same as the input has; if rtype is negative, the output matrix will have the same type as the input.
- **alpha** – optional scale factor.
- **beta** – optional delta added to the scaled values.

m(x,y) = <uchar>(this(x,y) *alpha + beta)

Convolution processing (컨벌루션 처리) 기반 필터링: 에지

- 수행 결과



Convolution processing (컨벌루션 처리) 기반 필터링

2차 미분 필터

- Laplacian filter
  - void **Laplacian**(InputArray **src**, OutputArray **dst**, int **ddepth**, int **ksize**=1, double **scale**=1, double **delta**=0, int **borderType**=BORDER_DEFAULT)

Parameters:	•**src** – Source image. •**dst** – Destination image of the same size and the same number of channels as src . •**ddepth** – Desired depth of the destination image. •**ksize** – Aperture size used to compute the second-derivative filters. See getDerivKernels() for details. The size must be positive and odd. •**scale** – Optional scale factor for the computed Laplacian values. By default, no scaling is applied. See getDerivKernels() for details. •**delta** – Optional delta value that is added to the results prior to storing them in dst . •**borderType** – Pixel extrapolation method. See borderInterpolate for details.

Convolution processing (컨벌루션 처리) 기반 필터링: 에지

- Laplacian 기본 예제

```
int main( int argc, char** argv )
{
        Mat src, src_gray, dst;
        int kernel_size = 3;
        int scale = 1;
        int delta = 0;
        int ddepth = CV_16S;
        char* window_name = "Laplace Demo";
        char* window_name1 = "Original Image";

        int c;

        /// Load an image
        src = imread("test.jpg",1);

        if( !src.data )
            { return -1; }

        namedWindow( window_name1, CV_WINDOW_AUTOSIZE );
        imshow(window_name1, src);

        /// Remove noise by blurring with a Gaussian filter
        GaussianBlur( src, src, Size(3,3), 0, 0, BORDER_DEFAULT );
```

(계속)

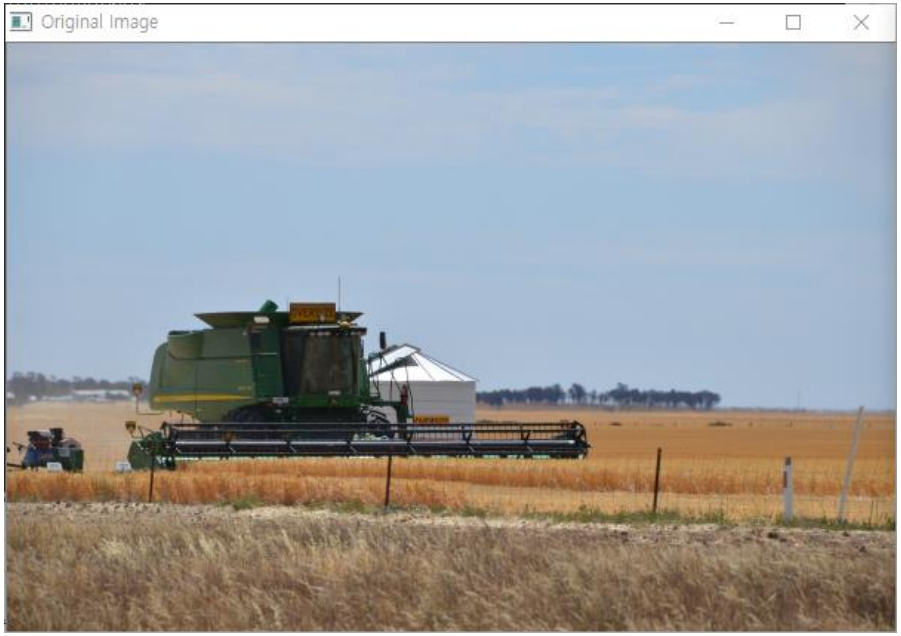
Convolution processing (컨벌루션 처리) 기반 필터링: 에지

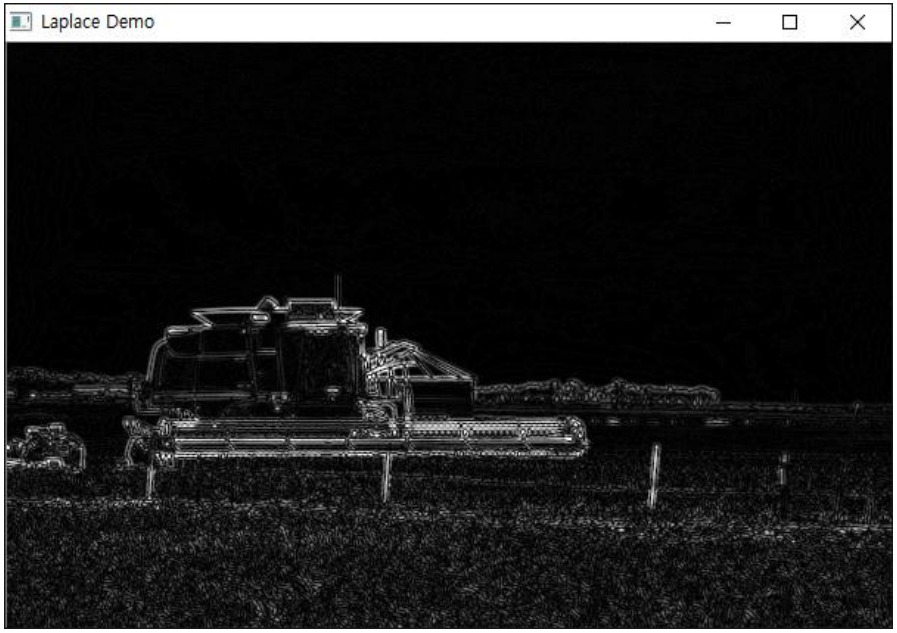
```
    /// Convert the image to grayscale
      cvtColor( src, src_gray, CV_BGR2GRAY );

      /// Create window
      namedWindow( window_name, CV_WINDOW_AUTOSIZE );

      /// Apply Laplace function
      Mat abs_dst;

      Laplacian( src_gray, dst, ddepth, kernel_size, scale, delta, BORDER_DEFAULT );
      convertScaleAbs( dst, abs_dst );

      /// Show what you got
      imshow( window_name, abs_dst );

      waitKey(0);

      return 0;
}
```

- **Void convertScaleAbs(InputArray src, OutputArray dst, double alpha=1, double beta=0) →** $\mathtt{dst}(I) = \mathtt{saturate\_cast<uchar>}(|\mathtt{src}(I) * \mathtt{alpha} + \mathtt{beta}|)$

Convolution processing (컨벌루션 처리) 기반 필터링: 에지

- 수행 결과





Convolution processing (컨벌루션 처리) 기반 필터링: 에지

- 컬러 영상 그대로 Laplacian 연산 처리

(계속)

```
/// Load an image
src = imread("test.jpg",1);

if( !src.data ){ return -1; }

namedWindow( window_name1, CV_WINDOW_AUTOSIZE );
imshow(window_name1, src);

/// Remove noise by blurring with a Gaussian filter
GaussianBlur( src, src, Size(3,3), 0, 0, BORDER_DEFAULT );

 /// Apply Laplace function
Mat abs_dst;

Laplacian( src, dst, ddepth, kernel_size, scale, delta, BORDER_DEFAULT );
convertScaleAbs( dst, abs_dst );

/// Show what you got
imshow( window_name, abs_dst );
```

(계속)

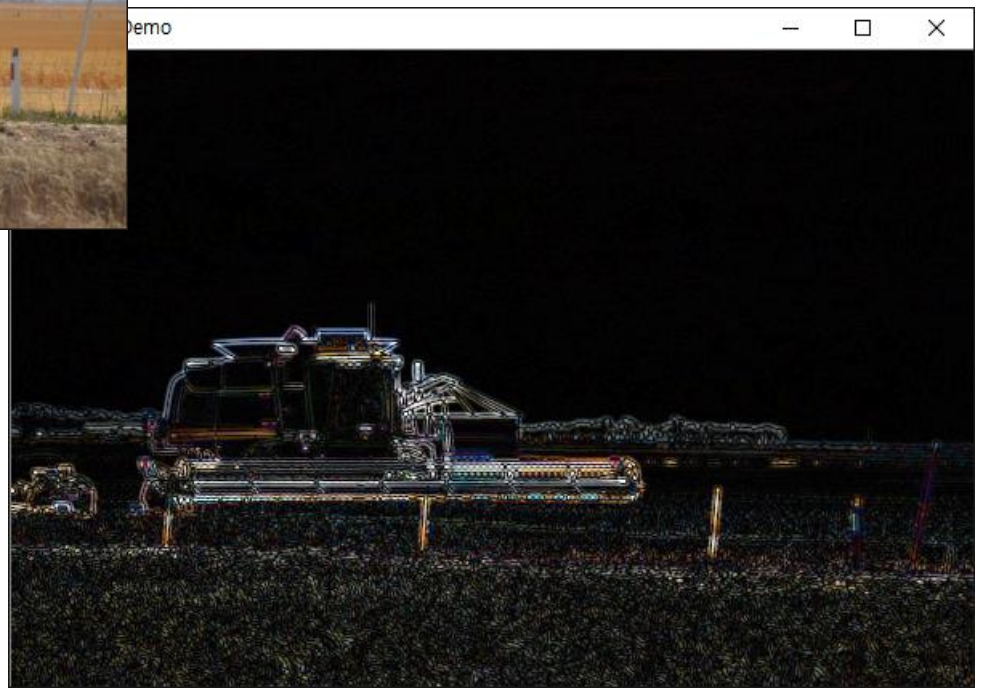
void **convertScaleAbs**(InputArray **src**, OutputArray **dst**, double **alpha**=1, double **beta**=0)

Parameters:
- **src** – input array.
- **dst** – output array.
- **alpha** – optional scale factor.
- **beta** – optional delta added to the scaled values

$$\texttt{dst}(I) = \texttt{saturate\_cast<uchar>}(|\texttt{src}(I) * \texttt{alpha} + \texttt{beta}|)$$

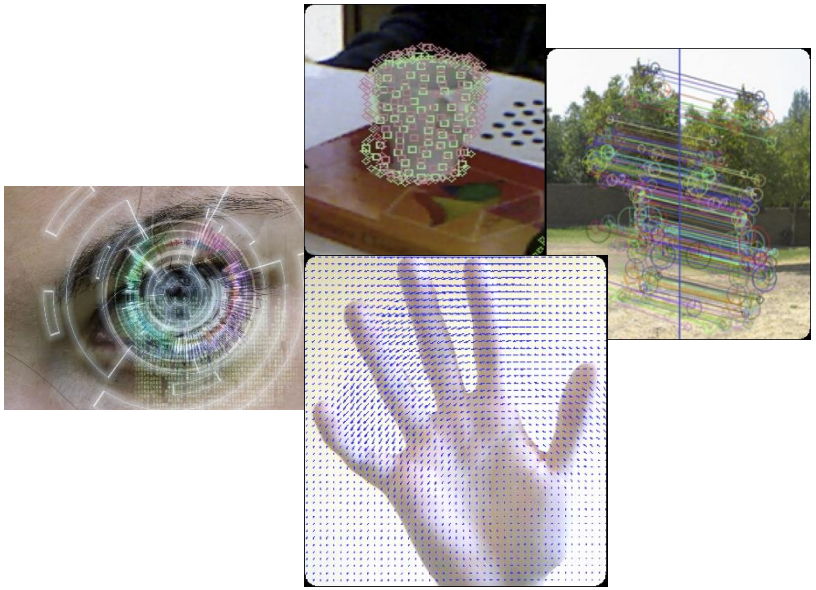
Convolution processing (컨벌루션 처리) 기반 필터링: 에지

- 수행 결과



COMPUTER VISION 비젼 프로그래밍

Thank you and Question?



IVPL
Intelligent Vision Processing Lab