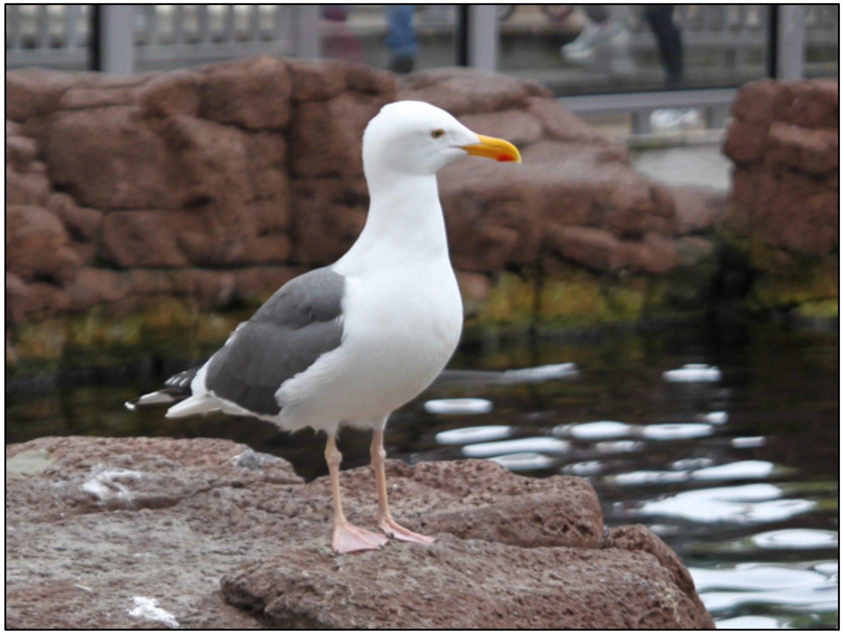
OpenCV-픽셀 다루기

김성영교수
금오공과대학교
컴퓨터공학과

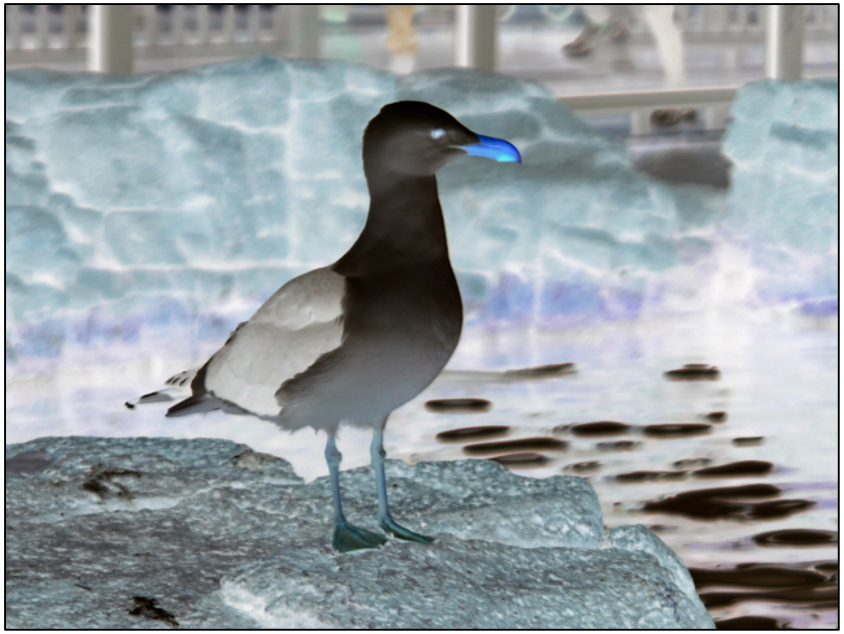
학습 내용

- 픽셀에 임의 접근하기
- 영상 스캐닝하기
- 방법에 따른 처리 속도 비교

Example: image Invert



Input image



Result image

$$value = 255 - value$$

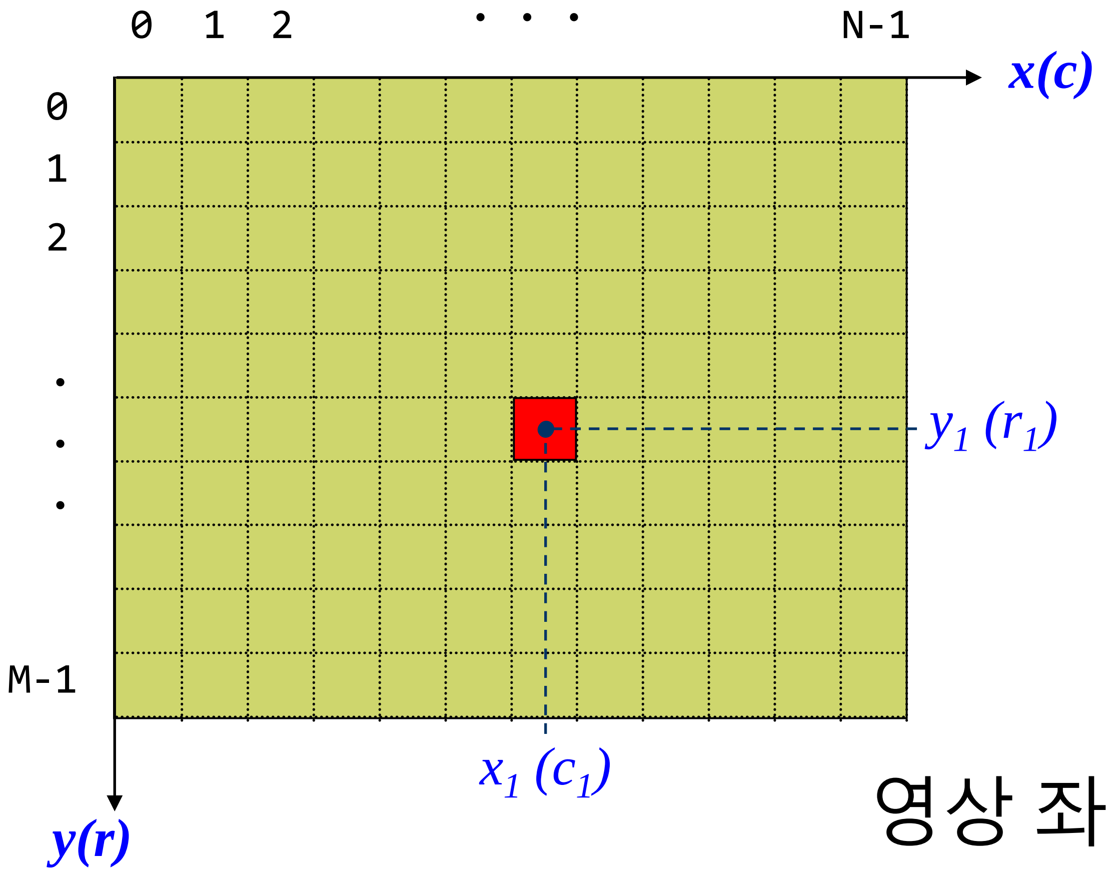
```
int main( void )
{
   . . .

   Mat image = imread( . . . );

   . . .

   if( image.channels( ) == 3 )
      colorinvert( image );
   else if(image.channels( ) == 1 )
      grayscaleinvert( image );

   . . .
}
```



영상 좌표 ⇨ (x_1, y_1)

행렬 위치 ⇨ (r_1, c_1)

픽셀에 임의 접근하기: at( ) method

- `Mat::at<Datatype>( int row, int col )`
  - □ for grayscale images
    - image.at <uchar> (*r*, *c*) = *value*;
  - □ for true color images
    - image.at <Vec3b> (*r*, *c*) [*channel*] = *value*;
    - channel : 0, 1, 2 (각 B, G, R에 대응)

- `Vec<Type, Number>`
  - □ `typedef Vec<uchar, 3> Vec3b;`

grayscale image

v	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*

100개의 pixels ⇨ 100개의 elements

true color image

b	*g*	*r*	*b*	*g*	*r*	*b*	*g*	*r*	*b*	*g*	*r*	*b*	*g*	*r*	*b*	*g*	*r*	*b*	*g*

100개의 pixels ⇨ 300개의 elements

```
void colorinvert( Mat &image )
{
    int numOfLines  = image.rows;  // number of lines in the image
    int numOfPixels = image.cols;  // number of pixels per a line

    for( int r=0; r<numOfLines; r++ )
    {
        for( int c=0; c<numOfPixels; c++ )
        {
            Vec3b &vec = image.at<Vec3b>( r, c );
            vec[0] = 255 - vec[0];
            vec[1] = 255 - vec[1];
            vec[2] = 255 - vec[2];
        }
    }
}
```

```
void grayscaleinvert( Mat &image )
{
   int numOfLines  = image.rows;  // number of lines in the image
   int numOfPixels = image.cols;  // number of pixels per a line

   for( int r=0; r<numOfLines; r++ )
   {
      for( int c=0; c<numOfPixels; c++ )
      {
         uchar &value = image.at<uchar>( r, c );
         value = 255 – value;
      }
   }
}
```

픽셀에 임의 접근하기: operator ( ) method

- `Mat_ ( row, col )`
  - □ for grayscale image
    - Mat_ <uchar> *grayimage* = *image*;
    - grayimage(*r*, *c*) = *value*;
  - □ for true color image
    - Mat_ <Vec3b> *colorimage* = *image*;
    - colorimage(*r*, *c*)[*channel*] = *value*;
  - □ Provides exactly the same result as the `at` method

```
void colorinvert( Mat &image )
{
    int numOfLines  = image.rows;  // number of lines in the image
    int numOfPixels = image.cols;  // number of pixels per a line
    Mat_<Vec3b> cimage = image;

    for( int r=0; r<numOfLines; r++ )
    {
        for( int c=0; c<numOfPixels; c++ )
        {
            Vec3b &vec = cimage( r, c );
            vec[0] = 255 - vec[0];
            vec[1] = 255 - vec[1];
            vec[2] = 255 - vec[2];
        }
    }
}
```

```
void grayscaleinvert( Mat &image )
{
   int numOfLines  = image.rows;  // number of lines in the image
   int numOfPixels = image.cols;  // number of pixels per a line
   Mat_<uchar> gimage = image;

   for( int r=0; r<numOfLines; r++ )
   {
      for( int c=0; c<numOfPixels; c++ )
      {
         uchar &value = gimage( r, c );
         value = 255 – value;
      }
   }
}
```

영상 스캐닝: ptr( ) method

- `Mat::ptr<Datatype>( int row )`
  - □ for grayscale images
    - uchar* data = image.ptr <uchar> (*r*);
  - □ for true color images
    - vec3b* data = image.ptr <vec3b> (*r*);
  - □ for grayscale & true color images
    - uchar* data = image.ptr <uchar> (*r*);

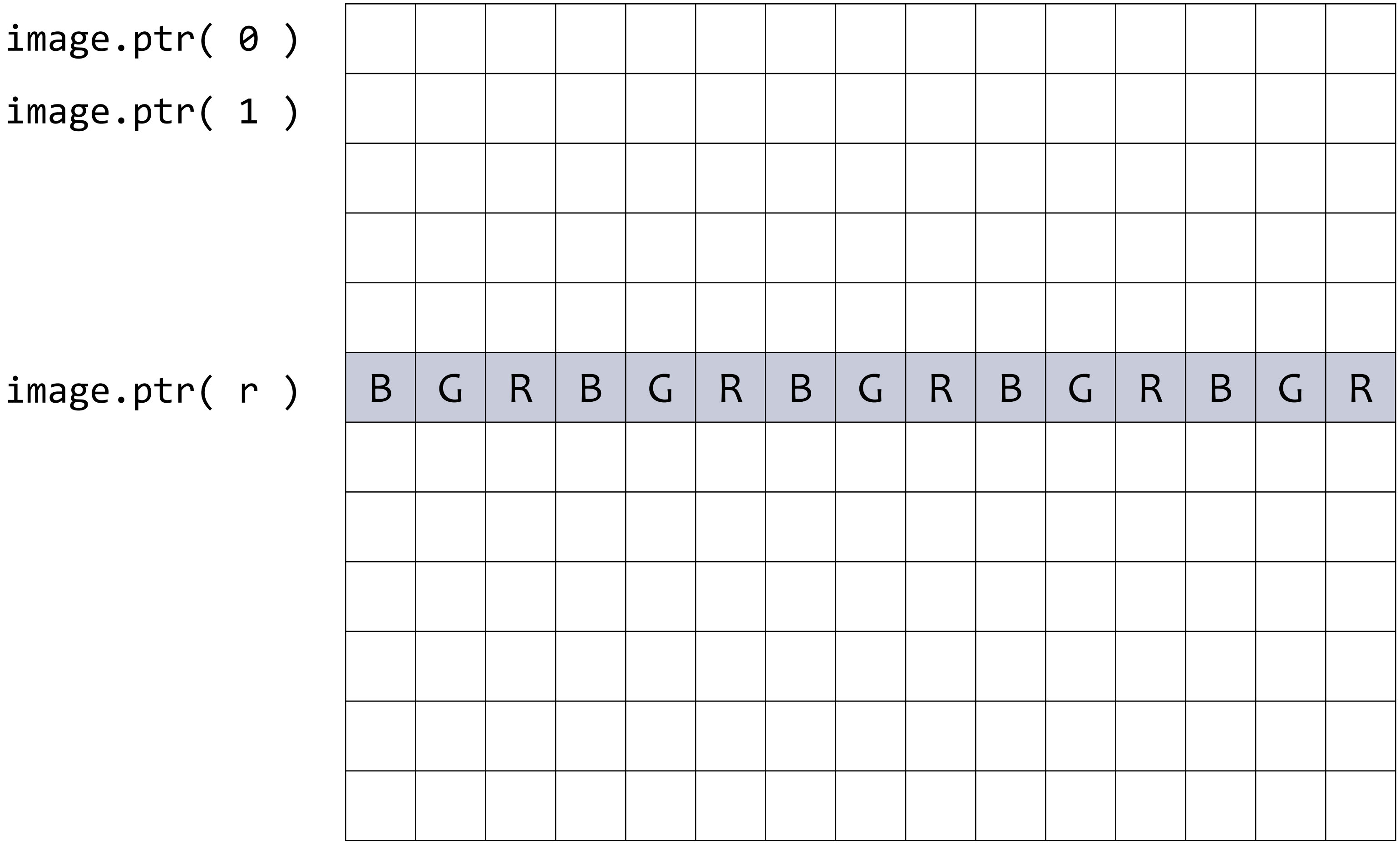


image.ptr(0)

image.ptr(1)

image.ptr(r)

B G R B G R B G R B G R B G R

```
vec3b* data = image.ptr <vec3b> ( r );
uchar* data = image.ptr <uchar> ( r );
```

```
void colorinvert( Mat &image )
{
    int numOfLines  = image.rows;  // number of lines in the image
    int numOfPixels = image.cols;  // number of pixels per a line

    for( int r=0; r<numOfLines; r++ )
    {
        Vec3b *data = image.ptr<Vec3b>( r );
        for( int c=0; c<numOfPixels; c++ )
        {
            data[c][0] = 255 - data[c][0];
            data[c][1] = 255 - data[c][1];
            data[c][2] = 255 - data[c][2];
        }
    }
}
```

```
void grayscaleinvert( Mat &image )
{
    int numOfLines  = image.rows;  // number of lines in the image
    int numOfPixels = image.cols;  // number of pixels per a line

    for( int r=0; r<numOfLines; r++ )
    {
        uchar *data = image.ptr<uchar>( r );
        for( int c=0; c<numOfPixels; c++ )
        {
            data[c] = 255 - data[c];
        }
    }
}
```

```
void invert( Mat &image )
{
    // number of lines in the image
    int numOfLines    = image.rows;
    // number of elements per a line
    int numOfElements = image.cols * image.channels();

    for( int r=0; r<numOfLines; r++ )
    {
        uchar *data = image.ptr<uchar>( r );
        for( int c=0; c<numOfElements; c++ )
        {
            data[c] = 255 - data[c];
        }
    }
}
```

fastest version !!

```
void colorinvert( Mat &image )
{
   int numOfLines  = image.rows;  // number of lines in the image
   int numOfPixels = image.cols;  // number of pixels per a line

   for( int r=0; r<numOfLines; r++ )
   {
      uchar *data = image.ptr<uchar>( r );
      for( int c=0; c<numOfPixels; c++ )
      {
         data[3*c+0] = 255 - data[3*c+0]; // *data++ = 255 - *data;
         data[3*c+1] = 255 - data[3*c+1]; // *data++ = 255 - *data;
         data[3*c+2] = 255 - data[3*c+2]; // *data++ = 255 - *data;
      }
   }
}
```

Input & Output argument 사용

- To Keep the original image intact
  - □ To create **a copy of the image** before calling the function
  - □ To give **the option** to the user to either use or not use the in-place processing

```
void colorinvert( const Mat &image,   // input image
                  Mat &result );      // output image
```

- `Mat::create( int rows, int cols, int type );`
  - □ Verify if the output image has an allocated data buffer with a size and pixel that match the ones of the input image

```
int main( void )
{
   . . .

   Mat image = imread( . . . );

   . . .

   Mat result;
   if( image.channels( ) == 3 )
      colorinvert( image, result );
      //colorinvert( image, image );
   else if(image.channels( ) == 1 )
      grayscaleinvert( image, result );
      //grayscaleinvert( image, image );

   . . .
}
```

```
void colorinvert( const Mat &image, Mat &result )
{
    int numOfLines  = image.rows;  // number of lines in the image
    int numOfPixels = image.cols;  // number of pixels per a line

    result.create( image.rows, image.cols, image.type() );

    for( int r=0; r<numOfLines; r++ )
    {
        const uchar *data_in = image.ptr<uchar>( r );
        uchar *data_out      = result.ptr<uchar>( r );
        for( int c=0; c<numOfPixels; c++ )
        {
            data_out[3*c+0] = 255 – data_in[3*c+0];
            data_out[3*c+1] = 255 – data_in[3*c+1];
            data_out[3*c+2] = 255 – data_in[3*c+2];
        }
    }
}
```

영상 스캐닝: iterators

- `MatIterator_<type> or Mat_<type>::iterator`
  - □ for grayscale images
    - MatIterator_<uchar> it;
    - Mat_<uchar>::iterator it;
  - □ for true color images
    - MatIterator_<Vec3b> it;
    - Mat_<Vec3b> it;

```
void colorinvert( const Mat &image, Mat &result )
{
   result.create( image.rows, image.cols, image.type() );

   // get iterators
   MatConstIterator_<Vec3b> in_it    = image.begin<Vec3b>();
   MatConstIterator_<Vec3b> in_itend = image.end<Vec3b>();
   MatIterator_<Vec3b> out_it    = result.begin<Vec3b>();
   MatIterator_<Vec3b> out_itend = result.end<Vec3b>();

   while( in_it!= in_itend )
   {
      (*out_it)[0]= 255 - (*in_it)[0];
      (*out_it)[1]= 255 - (*in_it)[1];
      (*out_it)[2]= 255 - (*in_it)[2];

      in_it++;
      out_it++;
   }
}
```

```
void grayscaleinvert( const Mat &image, Mat &result )
{
   result.create( image.rows, image.cols, image.type() );

   // get iterators
   MatConstIterator_<uchar> in_it    = image.begin<uchar>();
   MatConstIterator_<uchar> in_itend = image.end<uchar>();
   MatIterator_<uchar> out_it    = result.begin<uchar>();
   MatIterator_<uchar> out_itend = result.end<uchar>();

   while( in_it!= in_itend )
   {
      *out_it = 255 - *in_it;

      in_it++;
      out_it++;
   }
}
```

```
void colorinvert( const Mat &image, Mat &result )
{
   result.create( image.rows, image.cols, image.type() );

   // get iterators
   Mat_<Vec3b>::const_iterator in_it    = image.begin<Vec3b>();
   Mat_<Vec3b>::const_iterator in_itend = image.end<Vec3b>();
   Mat_<Vec3b>::iterator out_it    = result.begin<Vec3b>();
   Mat_<Vec3b>::iterator out_itend = result.end<Vec3b>();

   while( in_it!= in_itend )
   {
      (*out_it)[0] = 255 - (*in_it)[0];
      (*out_it)[1] = 255 - (*in_it)[1];
      (*out_it)[2] = 255 - (*in_it)[2];

      in_it++;
      out_it++;
   }
}
```

```
void grayscaleinvert( const Mat &image, Mat &result )
{
    result.create( image.rows, image.cols, image.type() );

    // get iterators
    Mat_<Vec3b>::const_iterator in_it    = image.begin<Vec3b>();
    Mat_<Vec3b>::const_iterator in_itend = image.end<Vec3b>();
    Mat_<Vec3b>::iterator out_it    = result.begin<Vec3b>();
    Mat_<Vec3b>::iterator out_itend = result.end<Vec3b>();

    while( in_it!= in_itend )
    {
        *out_it = 255 - *in_it;

        in_it++;
        out_it++;
    }
}
```

영상 스캐닝: 저수준의 포인터 연산

Image data의 시작 위치 지정

```
uchar *data = image.data;
```

(r, c) 위치의 픽셀 접근 (gray-scale image)

```
value = data + r*image.step + c;
```

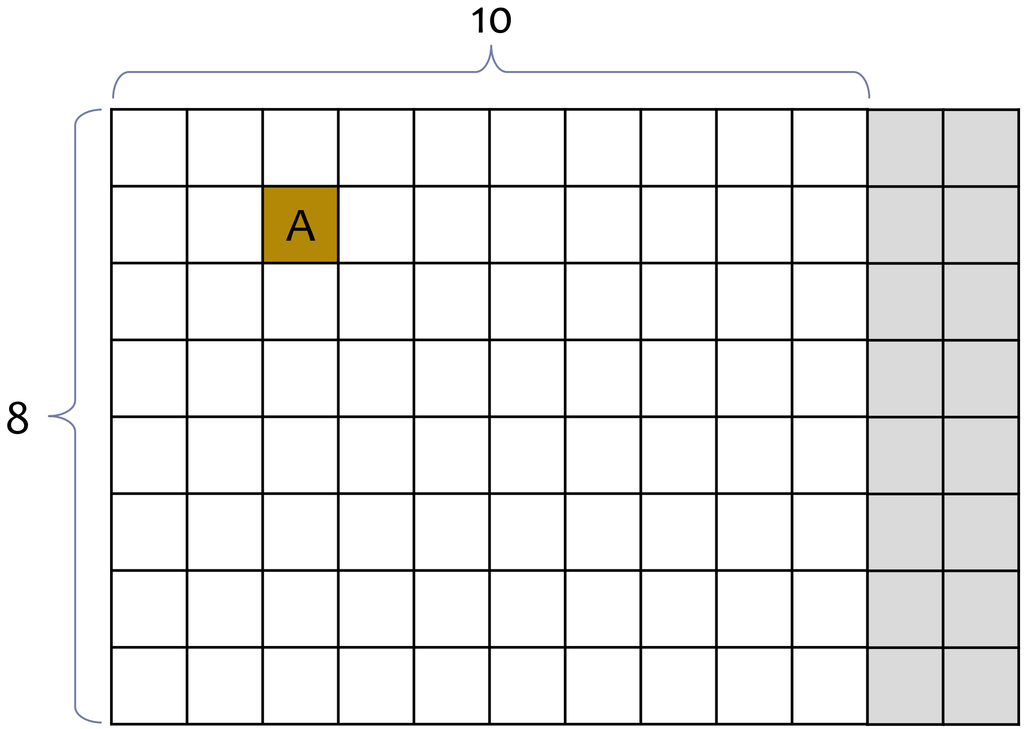
(r, c) 위치의 픽셀 접근 (true color image)

```
B_value = data + r*image.step + c*image.channels() + 0;
G_value = data + r*image.step + c*image.channels() + 1;
R_value = data + r*image.step + c*image.channels() + 2;
```

다음 라인(줄)으로 이동

```
data += image.step;
```

`Mat::isContinuous( )`

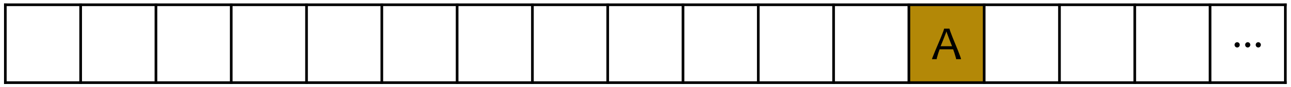


2차원 배열(data)

A ⇒ data[1][2]

1차원 배열(data)

A ⇒ data[12]



12 ⇨ 1*10 + 2 ⇨ r*cols + c

12 ⇨ 1*12 + 2 ⇨ r***step** + c

```
void colorinvert( const Mat &image, Mat &result )
{
    int numOfLines  = image.rows;  // number of lines in the image
    int numOfPixels = image.cols;  // number of pixels per a line

    result.create( image.rows, image.cols, image.type() );

    const uchar* data_in = image.data;
    uchar* data_out = result.data;
    for( int r=0; r<numOfLines; r++ )
    {
        for( int c=0; c<numOfPixels; c++ )
        {
            data_out[3*c + 0] = 255 - data_in[3*c + 0];
            data_out[3*c + 1] = 255 - data_in[3*c + 1];
            data_out[3*c + 2] = 255 - data_in[3*c + 2];
        }
        data_in  += image.step;
        data_out += result.step;
}
```

```
void grayscaleinvert( const Mat &image, Mat &result )
{
   int numOfLines  = image.rows;  // number of lines in the image
   int numOfPixels = image.cols;  // number of pixels per a line

   result.create( image.rows, image.cols, image.type() );

   const uchar* data_in = image.data;
   uchar* data_out = result.data;
   for( int r=0; r<numOfLines; r++ )
   {
      for( int c=0; c<numOfPixels; c++ )
      {
         data_out[c] = 255 - data_in[c];
      }
      data_in  += image.step;
      data_out += result.step;
}
```

처리 속도 비교

- at() method
  - true color image : 130.143
  - gray-scale image : 37.651
- operator () method
  - true color image : 148.016
  - gray-scale image : 73.1364
- ptr() method
  - true color image : 2.21535 (arguments: 2.61941)
  - gray-scale image : 1.04103 (arguments: 1.2792)
- iterator
  - true color image : 381.128
  - gray-scale image : 156.678

요약

- 픽셀 임의 접근 방법
  - Mat::at<Datatype>(int row, int col)
  - Mat_ (row, col)

- 영상 스캐닝 방법
  - Mat::ptr<Datatype>(int row)
  - MatIterator_<type> or Mat_<type>::iterator
  - Low level pointer operation

Reference

- R. Laganière, **OpenCV2 Computer Vision: Application Programming Cookbook,** PACKT Publishing, 2011
- G. Bradski and A. Kaebler, **Learning OpenCV: Computer Vision with the OpenCV Library**, O'REILLY, 2008
- 정성환, 이문호, **오픈소스 OpenCV를 이용한 컴퓨터 비전 실무 프로그래밍**, 홍릉과학출판사, 2007