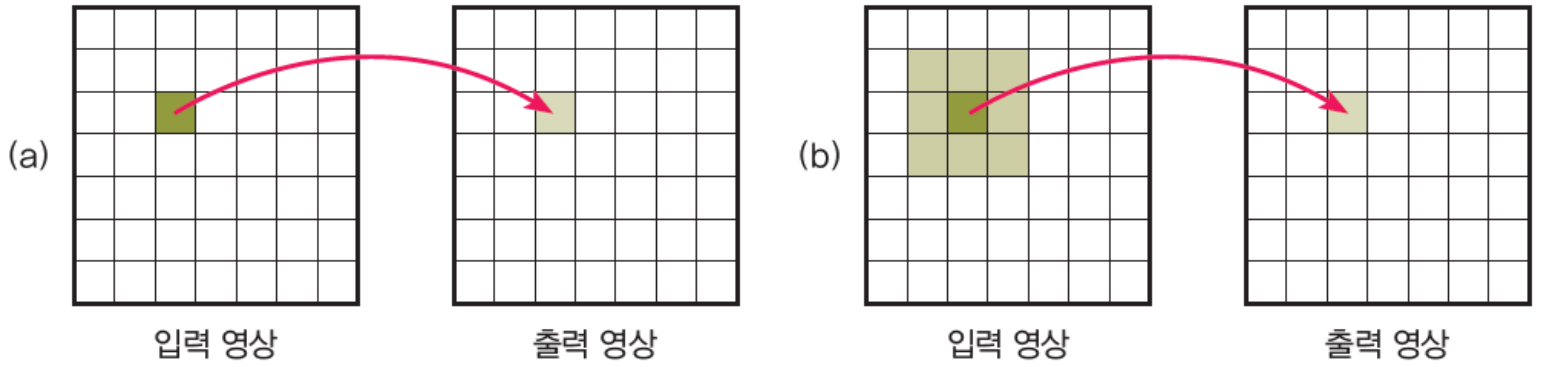
4장 화소처리

(Pixel Processing)

전통적인 영상 처리

- 화소 처리(point processing)
- 공간 필터링(filtering)



[그림 4.1] 화소 처리와 공간 필터링의 비교 (a) 화소 처리 (b) 공간 필터링

상수값에 의한 화소처리

- Image

50	60	60	60	60
50	60	70	70	70
60	60	70	70	70
70	70	70	80	80
80	70	80	80	90

- Operation

덧셈	OutImg[x][y] = InImg[x][y] + C1
뺄셈	OutImg[x][y] = InImg[x][y] – C2
곱셈	OutImg[x][y] = InImg[x][y] * C3
나눗셈	OutImg[x][y] = InImg[x][y] / C4

brightness

contrast

- Clipping

OutImg[x][y] = OutImg[x][y] >255 ? 255 : OutImg[x][y]
OutImg[x][y] = OutImg[x][y] < 0 ? 0 : OutImg[x][y]

상수값에 의한 화소처리

원영상

상수 더하기 (+ 60)

상수 곱하기 (* 1.4)

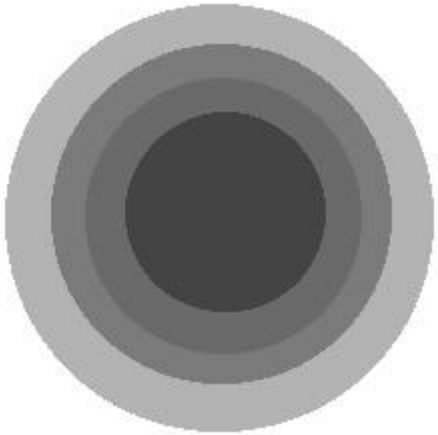
상수 빼기 (- 60)

상수 나누기 (/ 1.4)

두 영상 사이의 화소처리

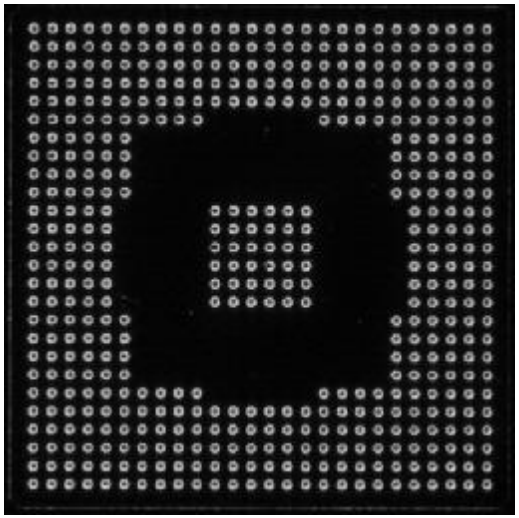
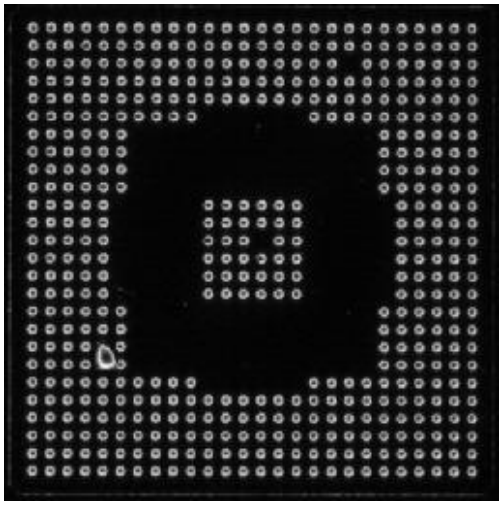
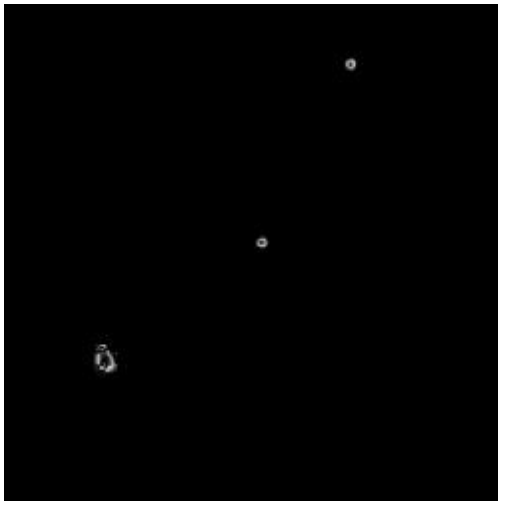
덧셈	OutImg[x][y] = InImg1[x][y] + InImg2[x][y]
뺄셈	OutImg[x][y] = InImg1[x][y] - InImg2[x][y]
곱셈	OutImg[x][y] = InImg1[x][y] * InImg2[x][y]
나눗셈	OutImg[x][y] = InImg1[x][y] / InImg2[x][y]

+

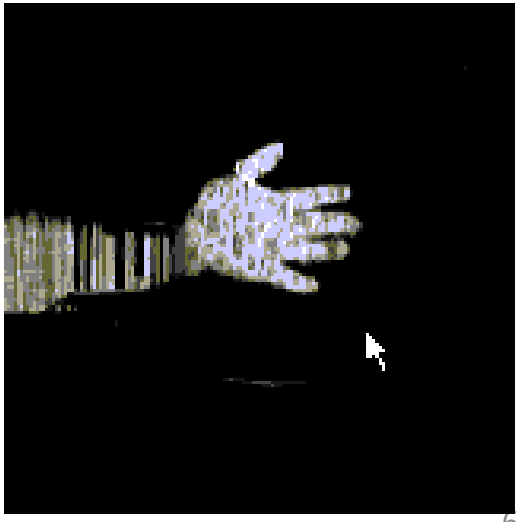


=

두 영상 사이의 화소처리

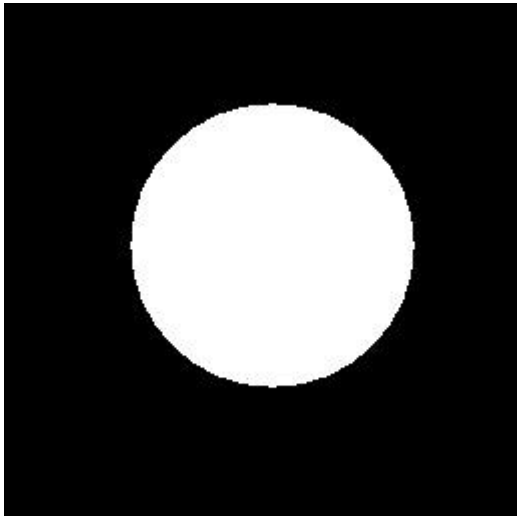








두 영상 사이의 화소처리

x



=

화소를 하나씩 처리하는 방법 #1

- Mat::at() 함수
  - 행렬의 지정된 원소(화소)에 접근하는 템플릿 함수
    - 템플릿 함수 - 모든 자료형 사용 가능
    - 반환되는 자료형으로 템플릿 구성도 가능

```
template <typename _Tp> _Tp& Mat::at(int i)
template <typename _Tp> _Tp& Mat::at(int i, int j)
template <typename _Tp> _Tp& Mat::at(int i, int j, int k)
template <typename _Tp> _Tp& Mat::at(Point pt)
template <typename _Tp> _Tp& Mat::at(const int* idx)
template <typename _Tp, int n> _Tp& Mat::at(const Vec<int, n>& idx)
```

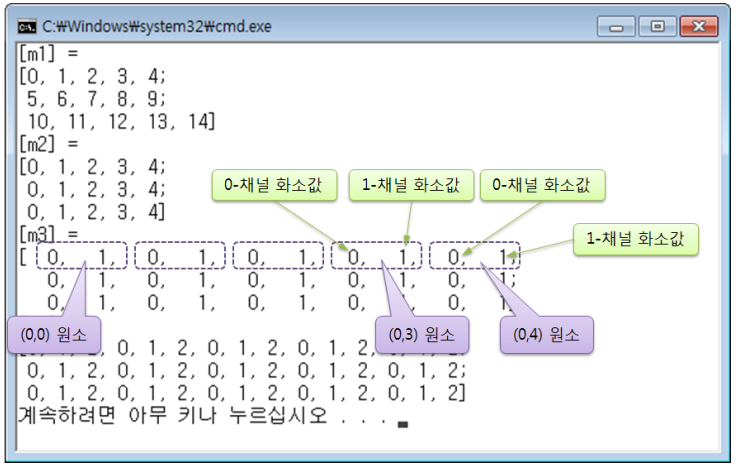
  - 예시

```
mat1.at<uchar>(10, 20);
mat2.at<int>(i, j);
mat3.at<double>(y, x);
mat4.at<Vec3d>(y, x)[0];
```

Mat::at() 함수

예제 6.1.1 Mat::at()을 통한 행렬 원소 접근 – mat_at.cpp

```
01  #include <opencv2/opencv.hpp>
02  using namespace cv;
03  using namespace std;
04  int main()
05  {
06      Mat  m1(3, 5, CV_32SC1);                    // 다양한 자료형의 행렬 선언
07      Mat  m2(3, 5, CV_32FC1);
08      Mat  m3(3, 5, CV_8UC2);
09      Mat  m4(3, 5, CV_32SC3);
10
11      for (int i = 0, k = 0; i < m1.rows; i++){   // 행렬 원소 순회 위한 반복문
12          for (int j = 0; j < m1.cols; j++, k++)
13          {
14              m1.at<int>(i, j) = k;
15              Point pt(j, i);
16              m2.at<float>(pt) = (float)j;        // Point로 행렬 원소 접근
17
18              int idx[2] = { i, j };
19              m3.at<Vec2b>(idx) = Vec2b(0 , 1);   // 배열로 행렬 원소 접근
20
21              m4.at<Vec3i>(i, j)[0] = 0;          // 배열첨자로 채널 원소 접근
22              m4.at<Vec3i>(i, j)[1] = 1;
23              m4.at<Vec3i>(i, j)[2] = 2;
24          }
25      }
26      cout << "[m1] = " << endl << m1 << endl;
27      cout << "[m2] = " << endl << m2 << endl;
28      cout << "[m3] = " << endl << m3 << endl;
29      cout << "[m4] = " << endl << m4 << endl;
30      return 0;
31  }
```

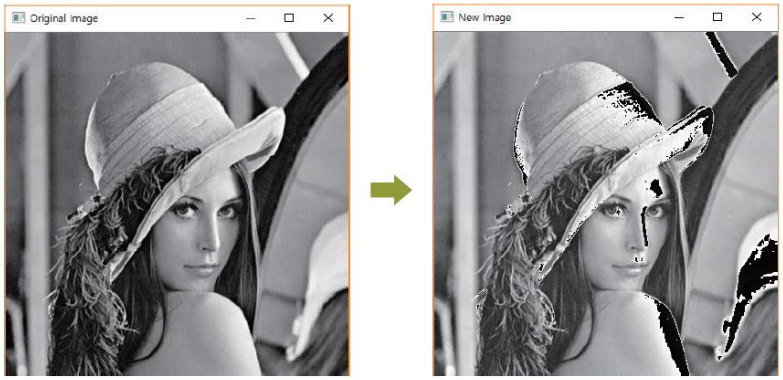


2채널 행렬이기에 반환자료형 Vec2b

3채널 행렬로 반환자료형은 Vec3i, 각 채널 원소 접근은 []로

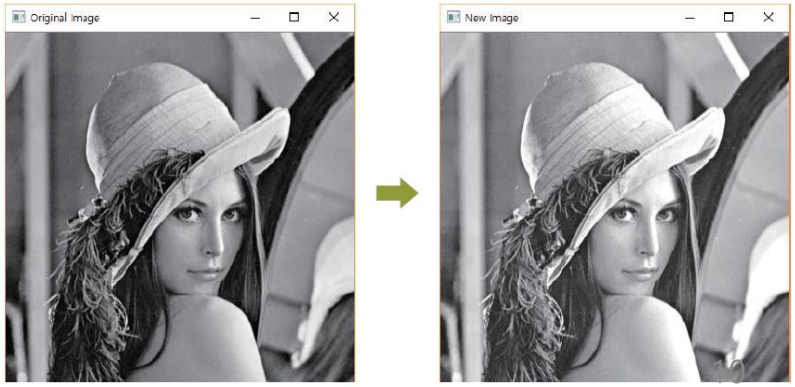
Mat::at() 함수

```
#include "opencv2/opencv.hpp"
using namespace cv;
using namespace std;

int main()
{
    Mat img = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Original Image", img);

    for (int r = 0; r < img.rows; r++)
        for (int c = 0; c < img.cols; ++c)
            img.at<uchar>(r, c) = img.at<uchar>(r, c) + 30;

    imshow("New Image", img);
    waitKey(0);
    return 0;
}
```



30이 더해지면 uchar 최대값인 255를 넘게 되어서 오버플로우 발생 가능

```
img.at<uchar>(r, c) =
saturate_cast<uchar>(img.at<uchar>(r, c) + 30);
```



Mat::at() 함수

- 함수로 만들기

```
#include "opencv2/opencv.hpp"
using namespace cv;
using namespace std;

void brighten(Mat& img, int value)
{
    for (int r = 0; r < img.rows; r++)
        for (int c = 0; c < img.cols; ++c)
                img.at<uchar>(r, c) = saturate_cast<uchar>(img.at<uchar>(r, c) + value);
}

int main()
{
    Mat img = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Original Image", img);

    brighten(img, 30);
    imshow("New Image", img);
    waitKey(0);

    return 0;
}
```

화소를 하나씩 처리하는 방법 #2

- Mat::ptr() 함수
  - C 스타일 연산자 [] 사용가능 => 빠른 처리 가능
  - 행 단위 처리

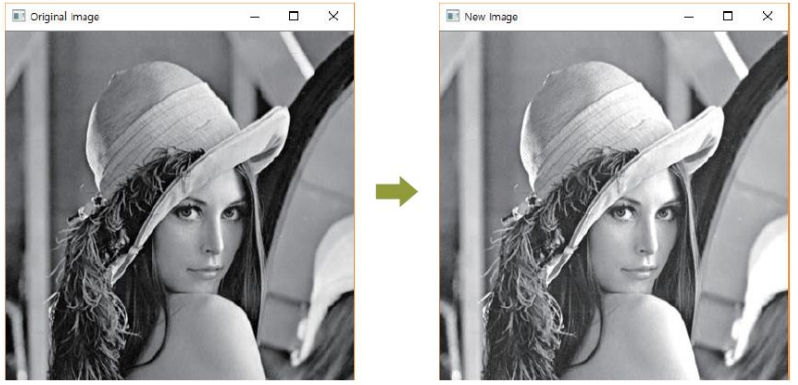
예제 6.1.2 Mat::ptr()을 통한 행렬 원소 접근 – mat_ptr.cpp

```
01  #include <opencv2/opencv.hpp>
02  using namespace cv;
03  using namespace std;
04  int main()
05  {
06      Mat  m1(3, 5, CV_8UC1);                          // uchar형 행렬 선언
07      Mat  m2(m1.size(), CV_32FC1);                    // float형 행렬 선언
08
09      for (int i = 0 , k =0 ; i < m1.rows; i++)
10      {
11          uchar * ptr_m1 = m1.ptr(i);                  // m1 행렬의 i행 첫 주소 반환
12          float * ptr_m2 = m2.ptr<float>(i);
13          for (int j = 0; j < m1.cols; j++)
14          {
15              ptr_m1[j] = j;
16              *(ptr_m2 + j) = (float)j;                // 포인터 접근 방식
17          }
18      }
19      cout << "m1 = " << endl << m1 << endl << endl;
20      cout << "m2 = " << endl << m2 << endl;
21      return 0;
22  }
```

```
C:₩Windows₩system32₩cmd.exe
m1 =
[  0,   1,   2,   3,   4;
   0,   1,   2,   3,   4;
   0,   1,   2,   3,   4]

m2 =
[0, 1, 2, 3, 4;
 0, 1, 2, 3, 4;
 0, 1, 2, 3, 4]
계속하려면 아무 키나 누르십시오 . . .
```

Mat::ptr() 함수

```
#include "opencv2/opencv.hpp"
using namespace cv;
using namespace std;

int main()
{
    Mat img = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Original Image", img);

    for (int r = 0; r < img.rows; r++) {
        uchar *p = img.ptr<uchar>(r);
        for (int c = 0; c < img.cols; ++c) {
                p[c] = saturate_cast<uchar>(p[c] + 30);
        }
    }
    imshow("New Image", img);
    waitKey(0);

return 0;
}
```



화소를 하나씩 처리하는 방법 #3

- Mat::convertTo 함수

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

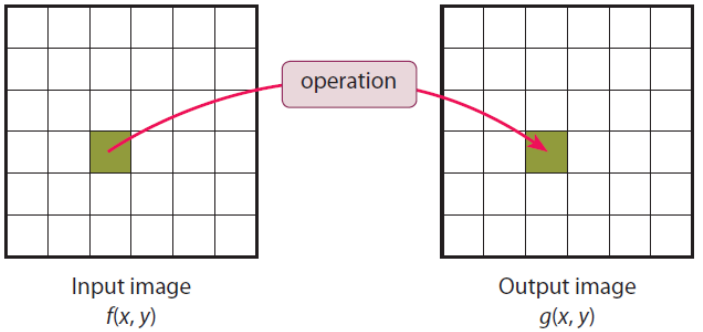
파라메터	설명
m	출력행렬
rtype	출력행렬 유형. 음수이면 원본과 동일유형
alpha	화소값에 곱해지는 수
beta	화소값에 더해지는 수

```
#include "opencv2/opencv.hpp"
using namespace cv;
using namespace std;

int main()
{
    Mat img = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Original Image", img);

    Mat oimage;
    img.convertTo(oimage, -1, 1, 30);

    imshow("New Image", oimage);
    waitKey(0);
    return 0;
}
```

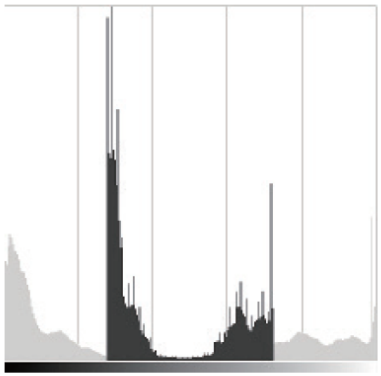
1을 곱하고 30을 더한다.

밝기 및 콘트라스트 조정

- $g(x, y) = \alpha \cdot f(x, y) + \beta$
  - α : contrast (대비) 조절
  - β : brightness (밝기) 조절



[그림 4.2] 밝은 회색은 원본 영상의 히스토그램, 어두운 회색은 보정된 영상의 히스토그램이다.



[그림 4.3] 밝은 회색은 원본 영상의 히스토그램, 어두운 회색은 콘트라스트가 줄어든 영상이다.

Mat::at() 사용

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace std;
using namespace cv;

int main()
{
    double alpha = 1.0;
    int beta = 0;
    Mat image = imread("d:/contrast.jpg");
    Mat oimage = Mat::zeros(image.size(), image.type());
    cout << "알파값을 입력하시오: [1.0-3.0]: "; cin >> alpha;
    cout << "베타값을 입력하시오: [0-100]: ";    cin >> beta;
    for (int y = 0; y < image.rows; y++) {
        for (int x = 0; x < image.cols; x++) {
            for (int c = 0; c < 3; c++) {
                    oimage.at<Vec3b>(y, x)[c] =
                      saturate_cast<uchar>(alpha*(image.at<Vec3b>(y, x)[c]) + beta);
            }
        }
    }
    imshow("Original Image", image);
    imshow("New Image", oimage);
    waitKey();
    return 0;
}
```

Mat::convertTo() 사용

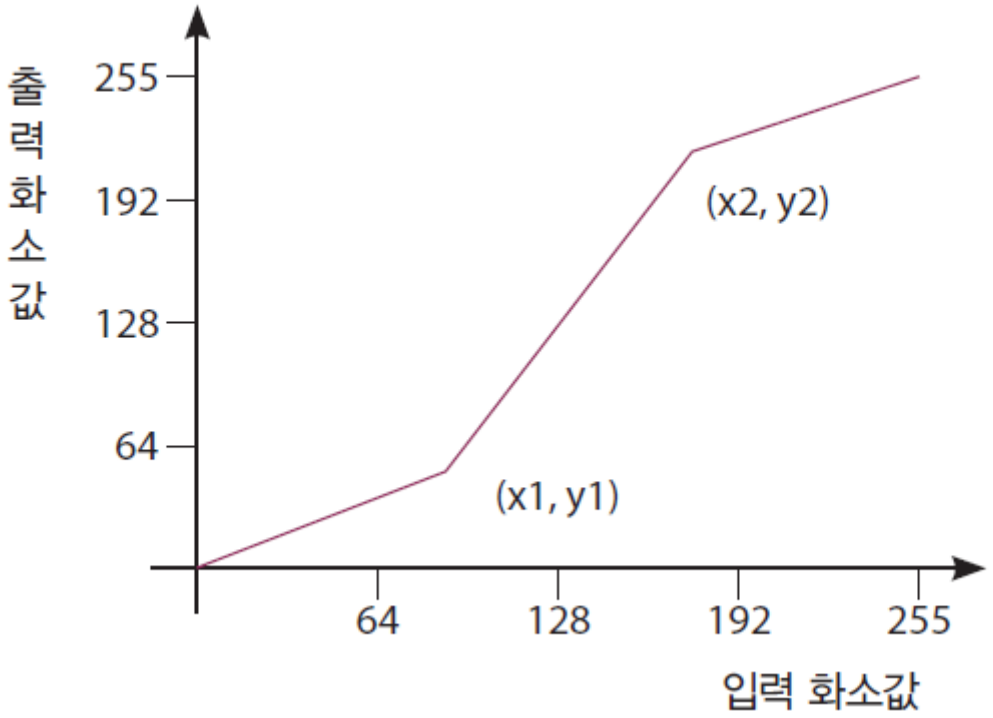
```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace std;
using namespace cv;

int main()
{
    double alpha = 1.0;
    int beta = 0;
    Mat image = imread("d:/contrast.jpg");
    Mat oimage;
    cout << "알파값을 입력하시오: [1.0-3.0]: "; cin >> alpha;
    cout << "베타값을 입력하시오: [0-100]: ";    cin >> beta;

    image.convertTo(oimage, -1, alpha, beta);
    imshow("Original Image", image);
    imshow("New Image", oimage);
    waitKey();
    return 0;
}
```

중복연산자 사용

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace std;
using namespace cv;

int main()
{
		double alpha = 1.0;
		int beta = 0;
		Mat image = imread("d:/contrast.jpg");
		Mat oimage;
		cout << "알파값을 입력하시오: [1.0-3.0]: "; cin >> alpha;
		cout << "베타값을 입력하시오: [0-100]: ";    cin >> beta;

		oimage = image * alpha + beta;
		imshow("Original Image", image);
		imshow("New Image", oimage);
		waitKey();
		return 0;
}
```

C++의 연산자 중복 기능을 이용하고 있다.

선형 콘트라스트 확대



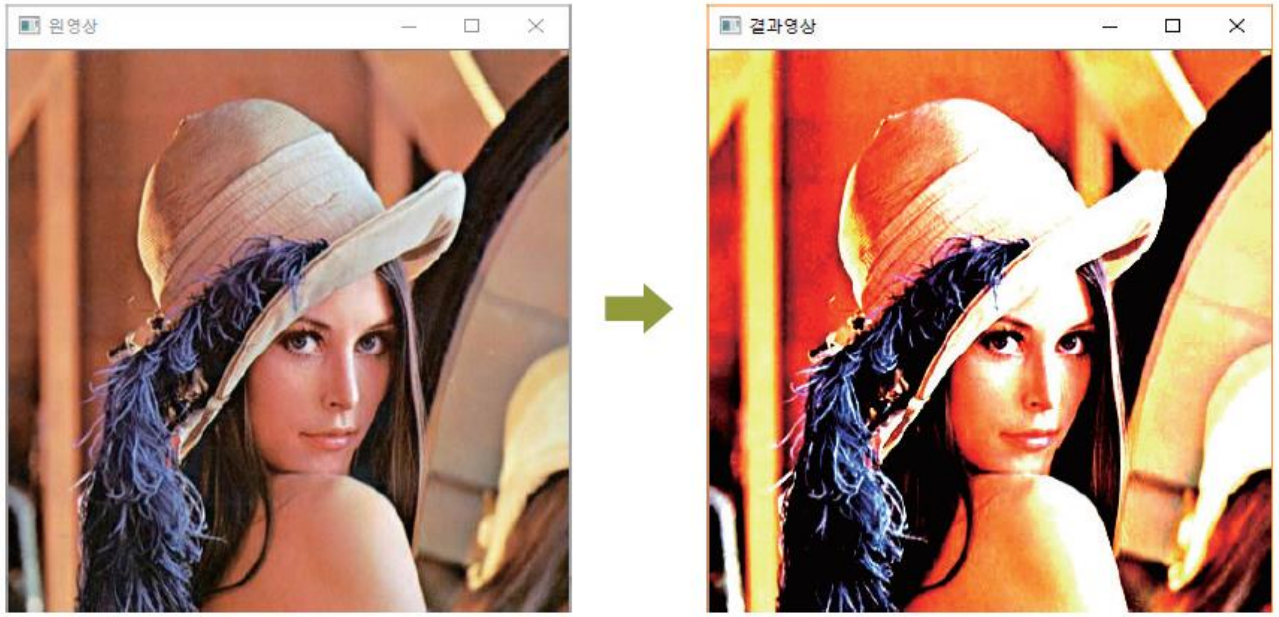
선형 콘트라스트 확대

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace std;
using namespace cv;

int contrastEnh(int input, int x1, int y1, int x2, int y2)
{
    double output;
    if (0 <= input && input <= x1) {
      output = y1 / x1 * input;
    }
    else if (x1 < input && input <= x2) {
      output = ((y2 - y1) / (x2 - x1)) * (input - x1) + y1;
    }
    else if (x2 < input && input <= 255) {
      output = ((255 - y2) / (255 - x2)) * (input - x2) + y2;
    }
    return (int)output;
}
```

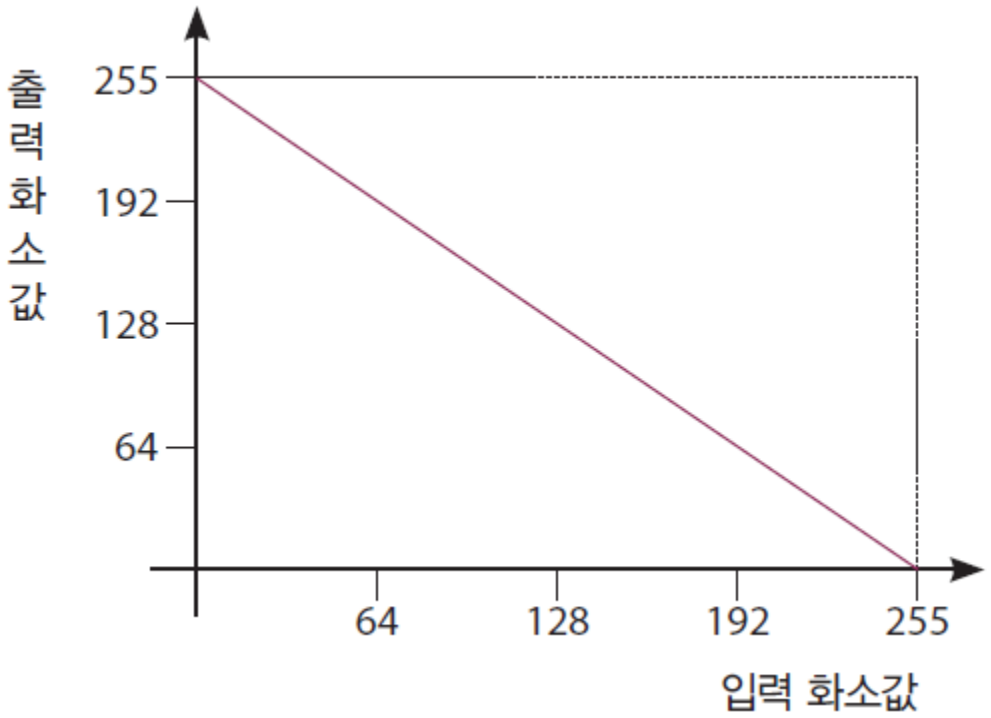
```
int main()
{
    Mat image = imread("d:/lenna.jpg");
    Mat oimage = image.clone();
    int x1, y1, x2, y2;
    cout << "x1 값을 입력하시오: "; cin >> x1;
    cout << "y1 값을 입력하시오: "; cin >> y1;
    cout << "x2 값을 입력하시오: "; cin >> x2;
    cout << "y2 값을 입력하시오: "; cin >> y2;

    for (int r = 0; r < image.rows; r++) {
      for (int c = 0; c < image.cols; c++) {
        for (int ch = 0; ch < 3; ch++) {
          int output = contrastEnh(image.at<Vec3b>(r, c)[ch], x1, y1, x2, y2);
          oimage.at<Vec3b>(r, c)[ch] = saturate_cast<uchar>(output);
        }
      }
    }
    imshow("원영상", image);
    imshow("결과영상", oimage);
    waitKey();
    return 0;
}
```

선형 콘트라스트 확대

- 실행결과



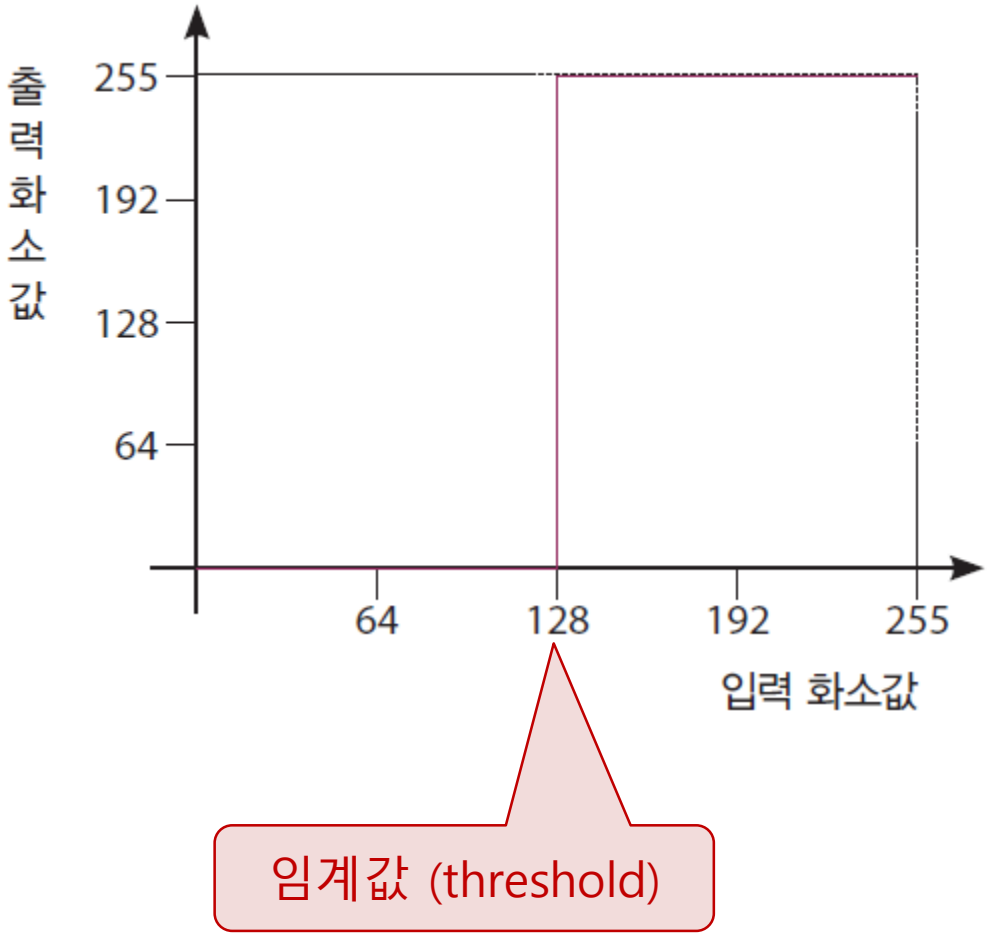
반전 (inversion)



반전

```
...
int main()
{
    Mat src;
    src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("원영상", src);

    Mat dst;
    dst = 255 - src;
    imshow("변경된 영상", dst);

    waitKey(0);
    return 0;
}
```

이진화 (binarization)



이진화

- threshold() 함수 : OpenCV 전역함수

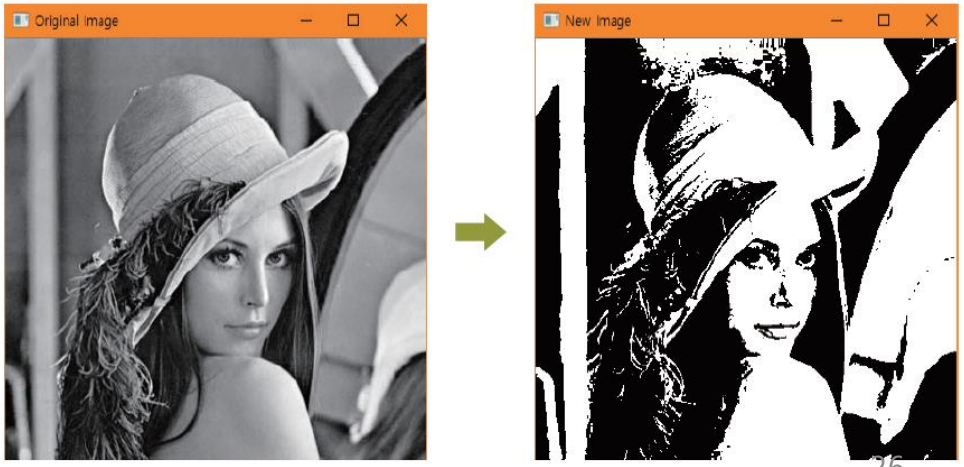
```
double threshold(InputArray src, OutputArray dst, double thresh,
        double maxval, int type)
```

매개 변수	설명
src	입력 영상. 1채널이어야 한다(8비트 또는 32-bit floating point).
dst	출력 영상
thresh	임계값
maxval	가능한 최대 출력값
type	이진화 종류. 우리는 THRESH_BINARY만 사용한다.

THRESH_BINARY	$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC	$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_TOZERO	$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_MASK	
THRESH_OTSU	flag, use Otsu algorithm to choose the optimal threshold value
THRESH_TRIANGLE	flag, use Triangle algorithm to choose the optimal threshold value

이진화

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace std;
using namespace cv;

int main()
{
    Mat image = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    Mat dst;
    int threshold_value = 127;
    threshold(image, dst, threshold_value, 255, THRESH_BINARY);
    imshow("Original Image", image);
    imshow("New Image", dst);
    waitKey(0);
    return 0;
}
```



이진화

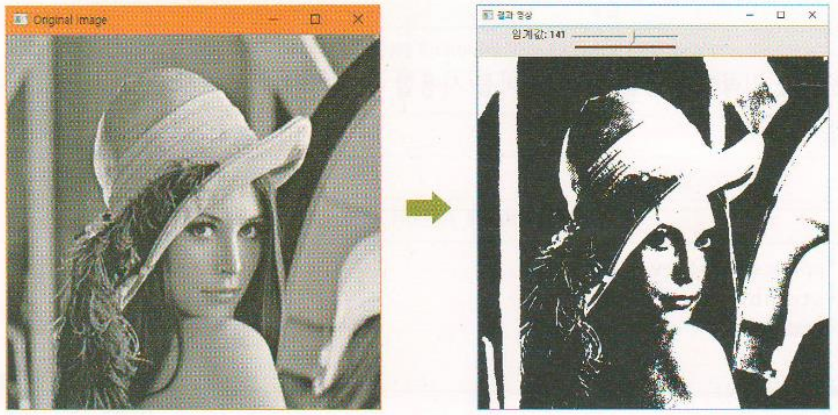
```
#include "opencv2/opencv.hpp"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;

Mat src, src_gray, dst;
int threshold_value = 0;
int threshold_type = 0;

void Threshold_Demo(int, void*)
{
   threshold(src_gray, dst, threshold_value, 255, threshold_type);
   imshow("결과 영상", dst);
}

int main()
{
   src = imread("d:/lenna.png");
   cvtColor(src, src_gray, CV_BGR2GRAY);
   namedWindow("결과 영상", CV_WINDOW_AUTOSIZE);

   createTrackbar("임계값", "결과 영상",
      &threshold_value,
      255, Threshold_Demo);

   Threshold_Demo(0, 0);

   while (true)
   {
      int c;
      c = waitKey(20);
      if ((char)c == 27) { // ESC 키가 입력되면 무한 루프 종료
         break;
      }
   }
   return 0;
}
```

트랙바를 사용하기 위하여 변수들을 전역 변수로 정의한다.

트랙바가 변경되면 이 함수가 호출된다.

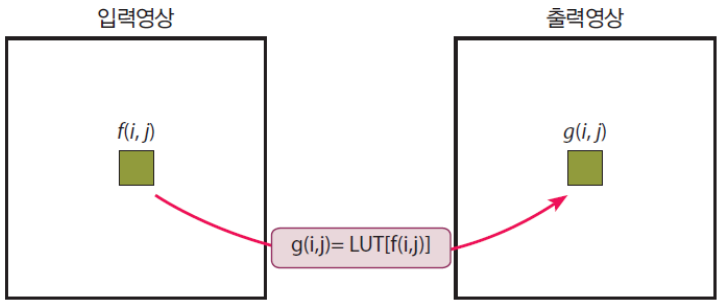
BGR 컬러 영상을 그레이스케일 영상으로 변환한다.

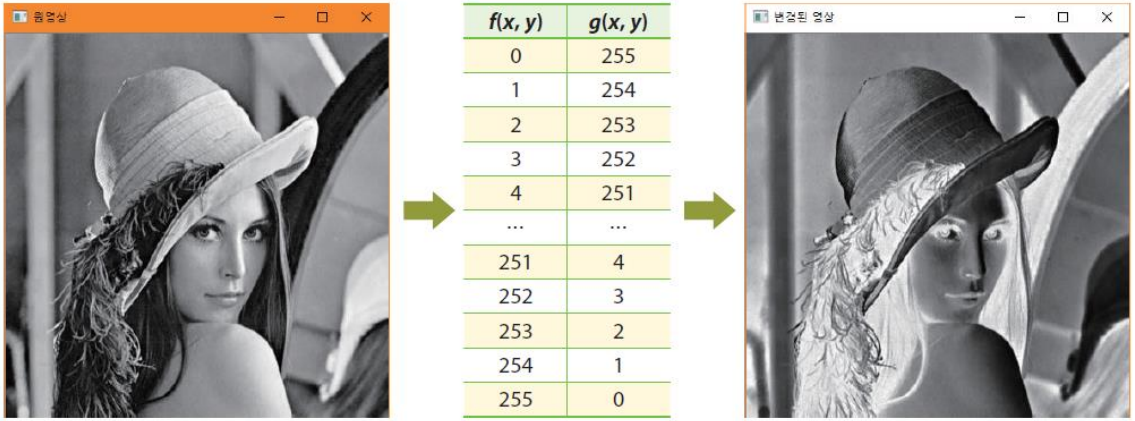
트랙바를 생성하고 콜백 함수와 연결한다.



LUT (Look-Up Table)

- Look-Up Table
  - 모든 화소값에 대한 출력값을 데이블에 미리 저장
  - 출력값 계산에 소요되는 시간 단축 (cf. Hash Table)





f(x, y)	g(x, y)
0	255
1	254
2	253
3	252
4	251
...	...
251	4
252	3
253	2
254	1
255	0

LUT

- LUT() 함수

LUT(input, table, output)

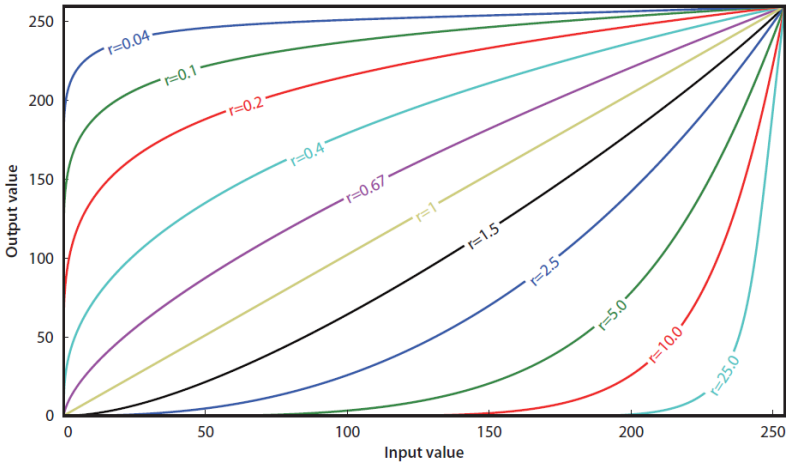
```
int main()
{
        Mat img1 = imread("d:/Lenna.jpg", IMREAD_GRAYSCALE);
        imshow("Original Image", img1);

        Mat table(1, 256, CV_8U);  // ①

        uchar* p = table.ptr();
        for (int i = 0; i < 256; ++i)
          p[i] = (i / 100) * 100;

        Mat img2;
        LUT(img1, table, img2);

        imshow("New Image", img2);
        waitKey(0);
        return 0;
}
```

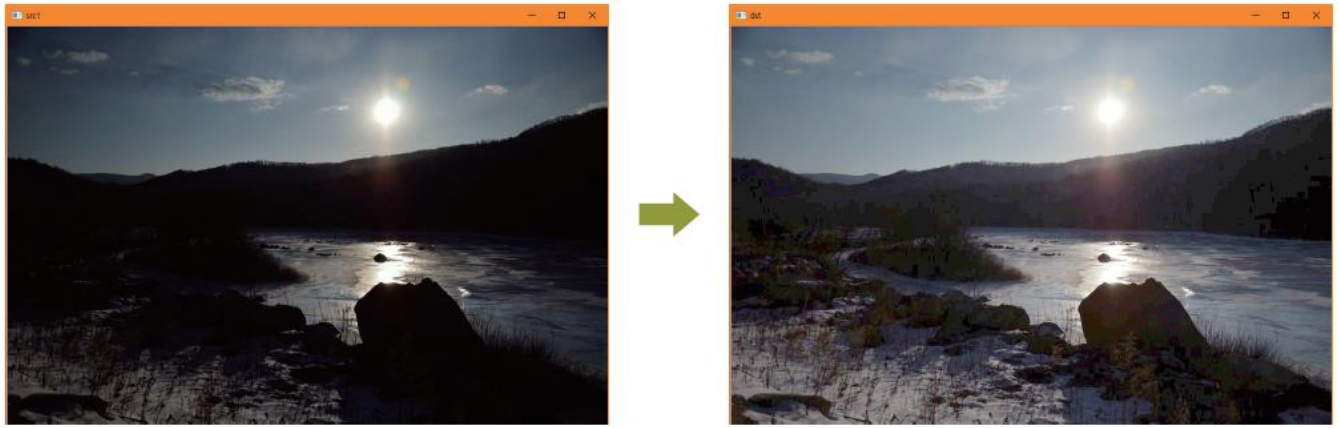
LUT를 초기화한다.

LUT를 적용한다.

감마보정

$$g(x, y) = \left(\frac{f(x, y)}{255} \right)^{\gamma} \times 255$$





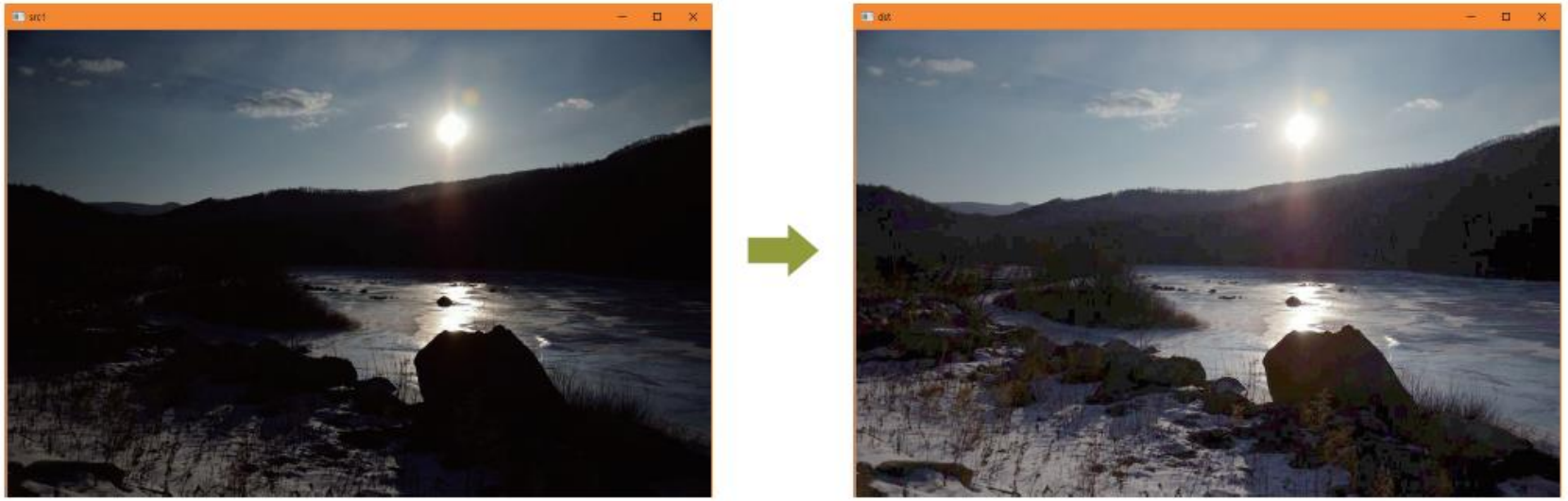
[그림 4.8] 감마 보정의 효과

감마보정

```
int main()
{
    Mat src1, src2, dst;
    double gamma = 0.5;

    src1 = imread("d:/gamma1.jpg");
    if (src1.empty()) { cout << "영상을 읽을 수 없습니다." << endl; return -1; }

    Mat table(1, 256, CV_8U);
    uchar * p = table.ptr();
    for (int i = 0; i < 256; ++i)
        p[i] = saturate_cast <uchar> (pow(i / 255.0, gamma) * 255.0);

    LUT(src1, table, dst);
    imshow("src1", src1);
    imshow("dst", dst);
    waitKey(0);
    return 0;
}
```

감마 변환을 수행하는 룩업 테이블을 생성한다.

실행결과



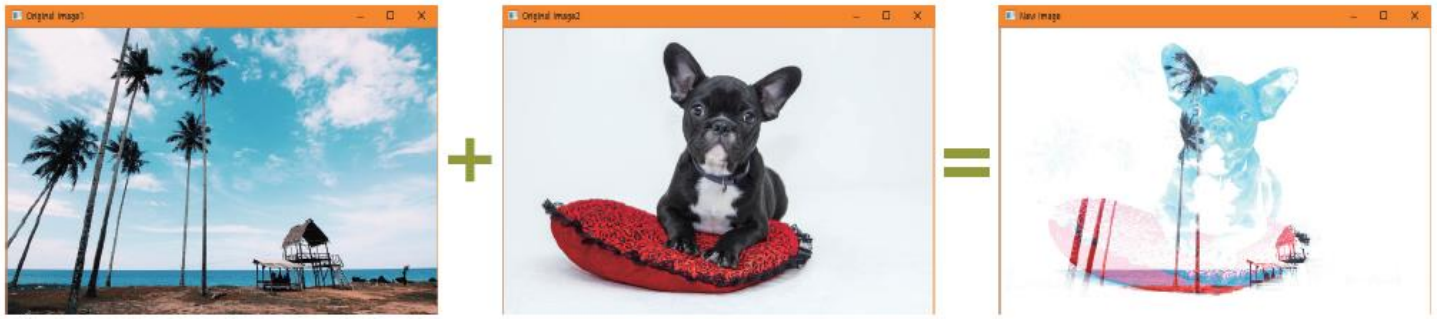
[그림 4.8] 감마 보정의 효과

(출처: https://www.codepool.biz/image-processing-opencv-gamma-correction.html)

영상합성

$$g(x, y) = f_1(x, y) + f_2(x, y)$$

```
int main()
{
        Mat src1 = imread("d:/test1.jpg");
        Mat src2 = imread("d:/test2.jpg");
        Mat dst;
        dst = src1 + src2;
        imshow("Original Image1", src1);
        imshow("Original Image2", src2);
        imshow("New Image", dst);
        waitKey(0);
        return 0;
}
```



선형영상합성

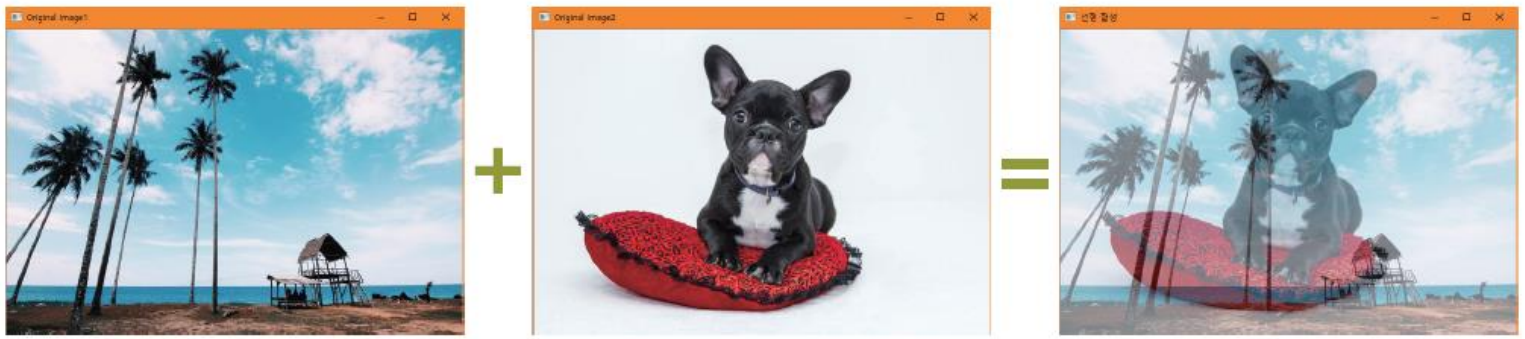
$$g(x, y) = (1 - \alpha) * f_1(x, y) + \alpha * f_2(x, y)$$

```
void addWeighted(InputArray src1, double alpha, InputArray src2,
        double beta, double gamma,   OutputArray dst, int dtype=-1)
```

매개 변수	설명
src1	첫 번째 입력 영상
alpha	첫 번째 영상의 가중치
src2	두 번째 입력 영상
beta	두 번째 영상의 가중치
gamma	화소의 합계에 더해지는 값
dst	출력 영상

선형영상합성

```
int main()
{
    double alpha = 0.5; double beta; double input;
    Mat src1, src2, dst;
    cout << "알파값을 입력하시오[0.0-1.0]: ";
    cin >> alpha;
    src1 = imread("d:/test1.jpg");
    src2 = imread("d:/test2.jpg");
    if (src1.empty()) { cout << "영상1을 로드할 수 없습니다." << endl; return -1; }
    if (src2.empty()) { cout << "영상2을 로드할 수 없습니다." << endl; return -1; }
    beta = (1.0 - alpha);
    addWeighted(src1, alpha, src2, beta, 0.0, dst);
    imshow("Original Image1", src1);
    imshow("Original Image2", src2);
    imshow("선형 합성", dst);
    waitKey(0);
    return 0;
}
```



HW

1. 영상파일을 읽어서 화면에 표시하고, 마우스로 드래그한 영역을 반전(inversion)하는 프로그램을 작성하라

2. 두 개의 영상파일을 읽어서 addWeighted() 함수를 사용하여 두 영상을 합성하는 프로그램을 작성하라. 트랙바를 사용하여 알파값을 조정할 수 있도록 하라.