5장 히스토그램

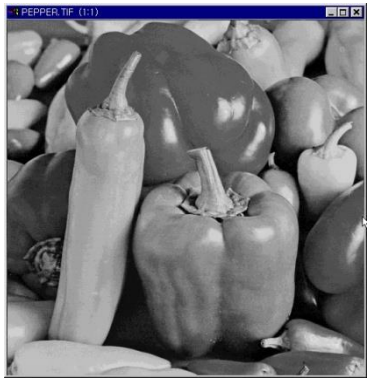
(Histogram)

히스토그램 (Histogram)

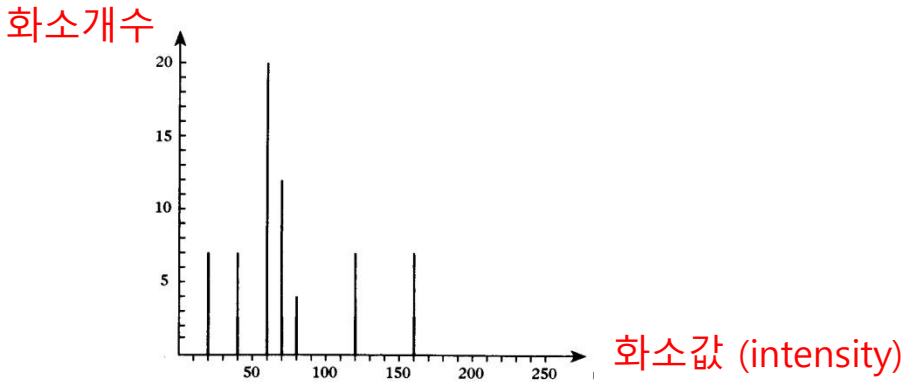
- Histogram
  - 영상의 밝기값에 대한 분포를 보여주는 그래프
  - X축: 밝기값(intensity), Y축: 빈도수(frequency)

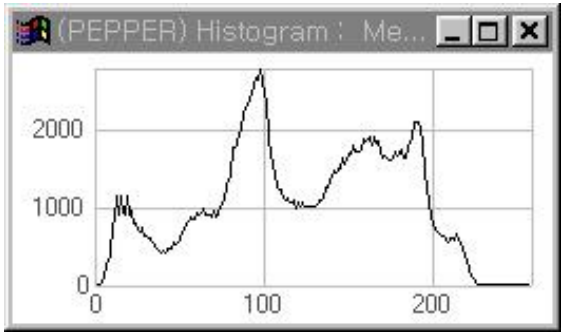
영상

20	20	20	20	20	20	20	40
160	60	60	60	60	60	60	40
160	60	70	70	70	70	60	40
160	60	70	80	80	70	60	40
160	60	70	80	80	70	60	40
160	60	70	70	70	70	60	40
160	60	60	60	60	60	60	40
160	120	120	120	120	120	120	120



히스토그램

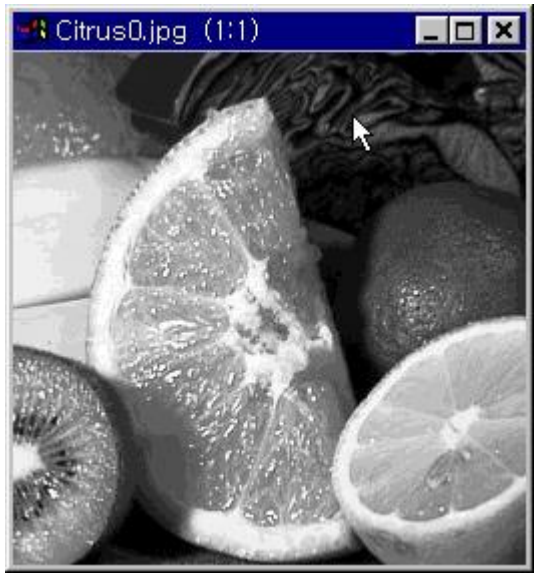


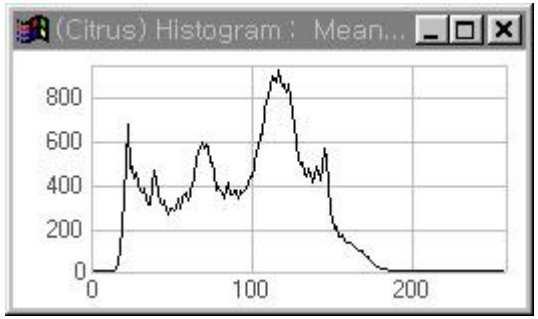


히스토그램


Citrus.jpg (1:1)


Citrus1.jpg (1:1)


Citrus0.jpg (1:1)


(Citrus) Histogram : Mean...
800
600
400
200
0
0
100
200


(Citrus1) Histogram : Mea...
1000
500
0
0
100
200

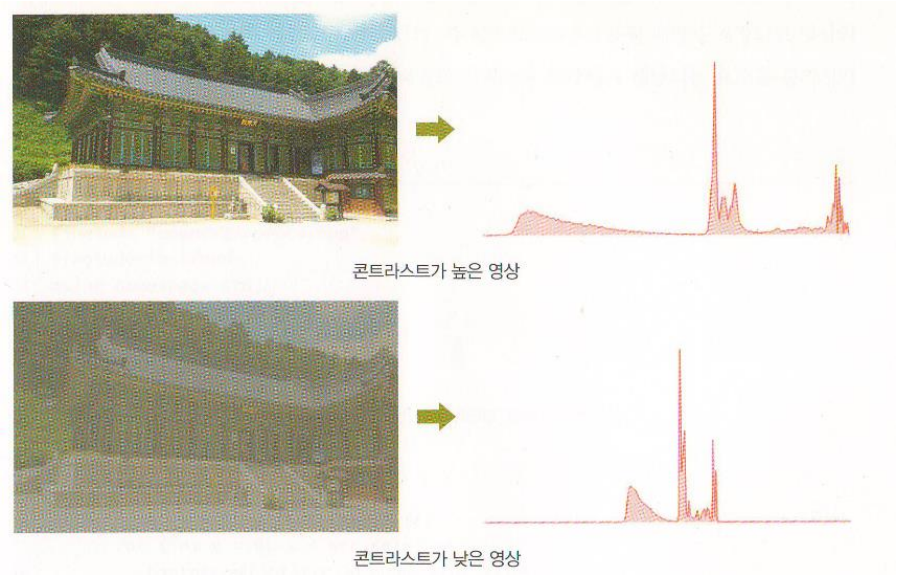

(Citrus0) Histogram : Mea...
400
300
200
100
0
0
100
200

그레이스케일 영상 히스토그램

컬러 영상 히스토그램

히스토그램으로 알 수 있는것

- 화소값들의 분포를 한눈에 볼 수 있다.

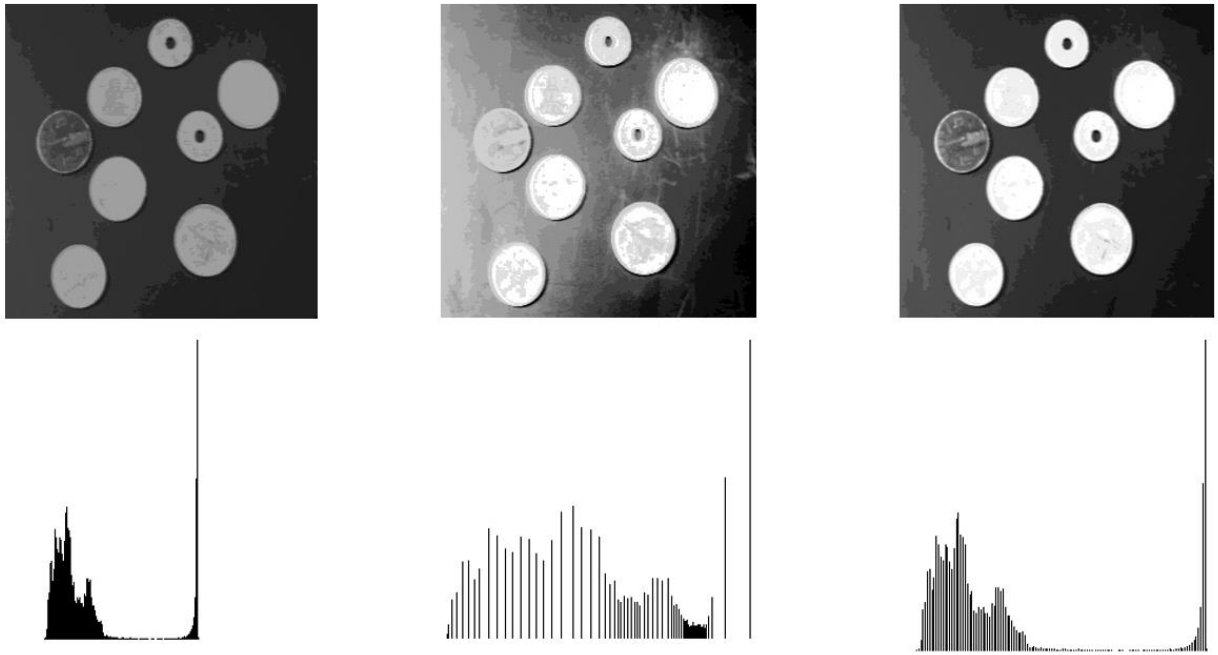


히스토그램의 용도

1. 화질 향상 (image enhancement)
   - 히스토그램 조작을 통한 가시도 향상
   - 사람의 눈은 밝기 (intensity) 보다 대비 (contrast) 에 민감

   1) 히스토그램 스트레칭 (histogram stretching)
   2) 히스토그램 평활화 (histogram equalization)

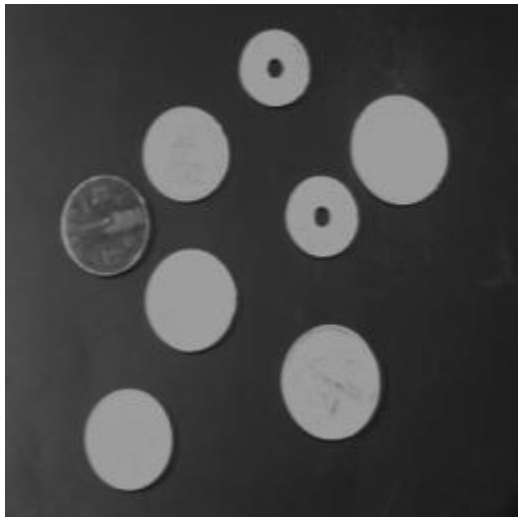
(원영상) (평활화 영상) (스트레칭 영상)



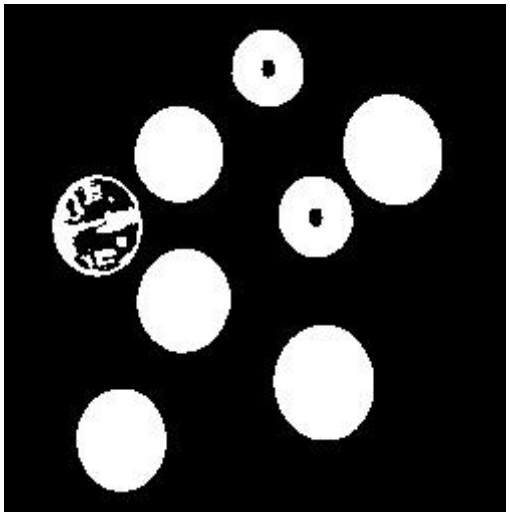
히스토그램의 용도

2. 배경과 물체 분리
   - 영상 이치화 (Binarization) 를 통한 물체 분리
   - 원영상: 영상의 밝기값 = 0~255
     이치화 영상: 영상의 밝기값 = 0 or 255
   - 공장자동화 용 물체인식/검사에 많이 응용됨

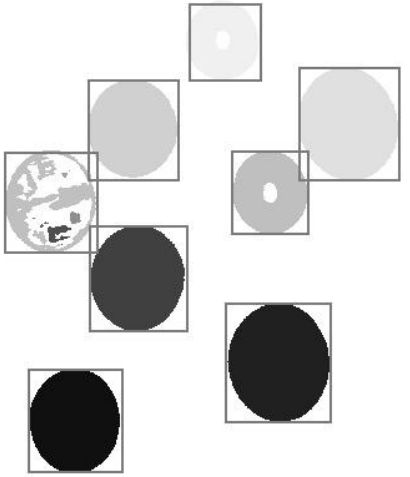
(원영상)



(이치화된 영상)



(물체인식)



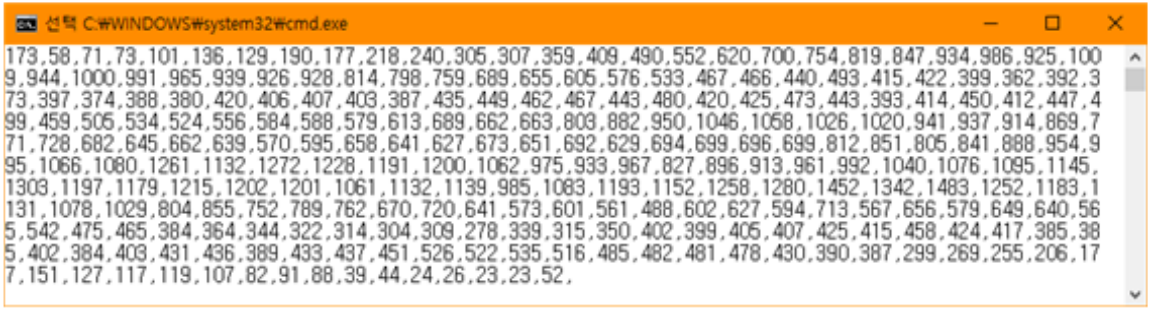
히스토그램 계산하기

- 히스토그램 알고리즘

```
for each pixel of the image
        value = intensity(pixel)
        histogram[value]++
end
```

히스토그램 계산하기

```
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Input Image", src);
    int histogram[256] = { 0 };

    for (int y = 0; y <src.rows; y++)
        for (int x = 0; x < src.cols; x++)
            histogram[(int)src.at<uchar>(y, x)]++;

    for (int count : histogram)
        cout << count << ",";
    waitKey(0);
    return 0;
}
```

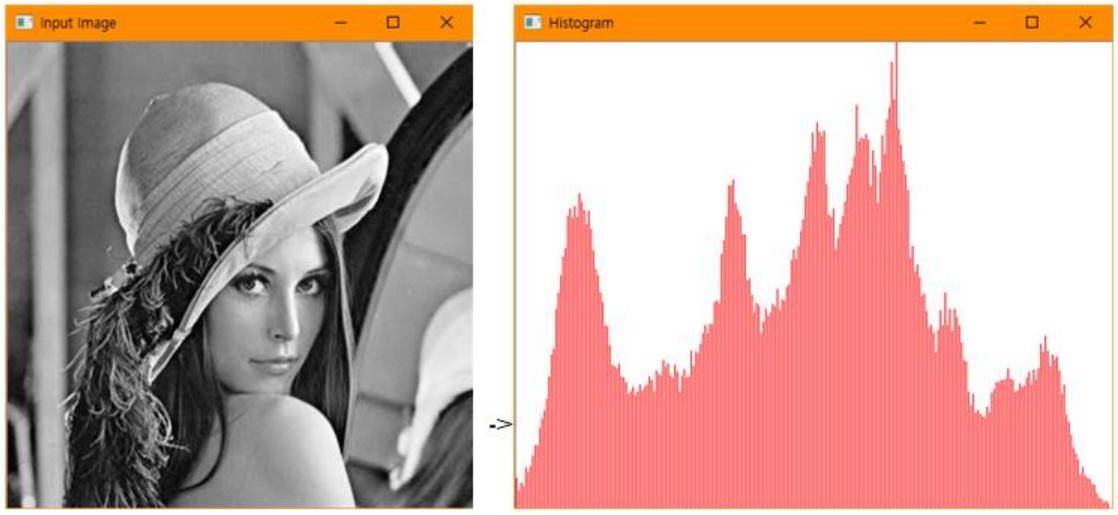


히스토그램 그리기

```
// 히스토그램을 받아서 막대그래프로 그린다.
void drawHist(int histogram[])
{
    int hist_w = 512;          // 히스토그램 영상의 폭
    int hist_h = 400;          // 히스토그램 영상의 높이
    int bin_w = cvRound((double)hist_w / 256);          // 빈의 폭

    // 히스토그램이 그려지는 영상(컬러로 정의)
    Mat histImage(hist_h, hist_w, CV_8UC3, Scalar(255, 255, 255));

    // 히스토그램에서 최대값을 찾는다.
    int max = histogram[0];
    for (int i = 1; i < 256; i++) {
        if (max < histogram[i])
                max = histogram[i];
    }
    // 히스토그램 배열을 최대값으로 정규화한다(최대값이 최대 높이가 되도록).
    for (int i = 0; i < 255; i++) {
        histogram[i] = floor(((double)histogram[i] / max)*histImage.rows);
    }

    // 히스토그램의 값을 빨강색 막대로 그린다.
    for (int i = 0; i < 255; i++) {
        line(histImage, Point(bin_w*(i), hist_h), Point(bin_w*(i), hist_h - histogram[i]),
                Scalar(0, 0, 255));
    }
    imshow("Histogram", histImage);
}
```

히스토그램 그리기

```
int main()
{
    Mat src = imread("lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Input Image", src);
    int histogram[256] = { 0 };

    for (int y = 0; y <src.rows; y++)
        for (int x = 0; x < src.cols; x++)
                histogram[(int)src.at<uchar>(y, x)]++;

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



OpenCV 함수로 히스토그램 계산하기

- calcHist()

```
void calcHist(const Mat* images, int nimages, const int* channels,
      InputArray mask, OutputArray hist, int dims, const int* histSize,
      const float** ranges, bool uniform=true, bool accumulate=false )
```

매개 변수	설명
images	소스 영상 행렬
nimages	소스 영상 행렬의 개수
channels	히스토그램을 계산하는데 사용되는 채널의 개수
mask	소스 배열에서 사용할 마스크 영상. 마스크 영상은 무시할 화소를 나타내며, 정의되지 않은 경우 사용되지 않는다.
hist	히스토그램이 저장될 Mat 객체
dims	히스토그램의 차원
histSize	사용된 차원당 상자[bin] 수
ranges	측정할 값의 범위
uniform	히스토그램이 균일한지를 표시하는 플래그
accumulate	true가 되면 히스토그램을 계산하기 전에 0으로 만들지 않는다. 즉 누적된다.

OpenCV 함수로 히스토그램 계산하기

- normalize()

```
normalize(b_hist, b_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
```

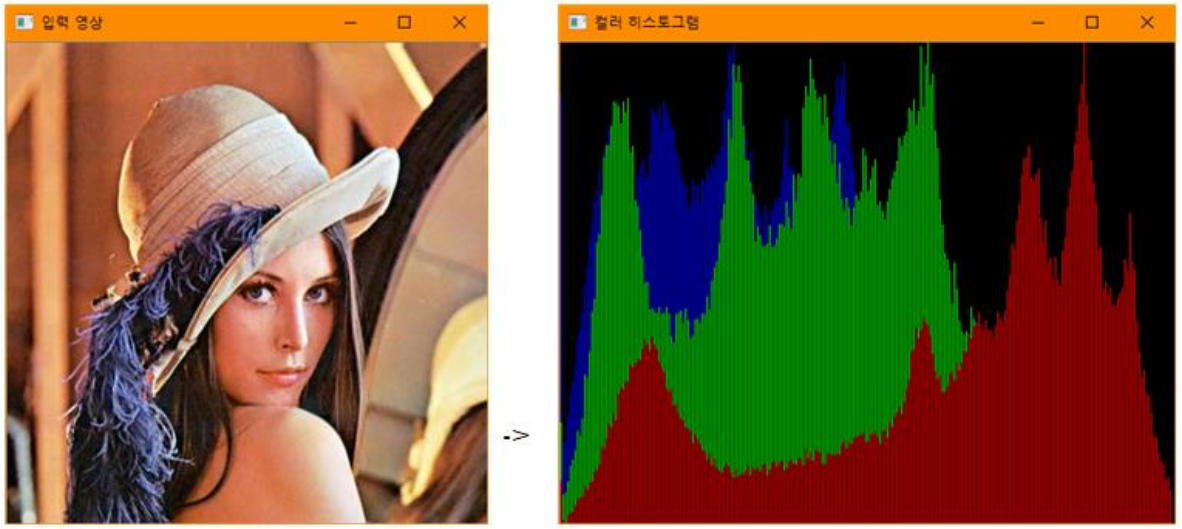
매개 변수	설명
b_hist	입력 배열
b_hist	정규화 되는 출력 배열. 현재는 동일하다.
0, histImage.rows	이 예제에서 히스토그램의 값을 정규화 하는데 사용되는 최소값과 최대값. 최소값은 0이고, 최대값은 히스토그램이 그려지는 영상의 세로길이(행의 개수)이다.
NORM_MINMAX	정규화의 종류 지정. 여기에서는 앞의 최소값과 최대값을 사용하여 정규화 한다.
-1	출력 영상은 입력 영상과 동일한 타입이라는 것을 나타낸다.
Mat()	마스크로 사용되는 Mat() 객체. 현재는 사용하지 않는다.

OpenCV 함수로 히스토그램 계산하기

```
int main(int argc, char** argv)
{
    Mat src = imread("d:/lenna.jpg",  IMREAD_COLOR);
    if (src.empty())  { return -1;}

    vector<Mat> bgr_planes;                 // 영상들의 벡터
    split(src, bgr_planes);                 // 입력 영상을 색상별로 분리한다.
    int histSize = 256;                     // 히스토그램에서 사용되는 상자의 개수
    float range[] = { 0, 256 };             // 화소값의 범위
    const float* histRange = { range };
    bool uniform = true, accumulate = false;

    Mat b_hist, g_hist, r_hist;
    calcHist(&bgr_planes[0], 1, 0, Mat(), b_hist, 1, &histSize, &histRange, uniform, accumulate);
    calcHist(&bgr_planes[1], 1, 0, Mat(), g_hist, 1, &histSize, &histRange, uniform, accumulate);
    calcHist(&bgr_planes[2], 1, 0, Mat(), r_hist, 1, &histSize, &histRange, uniform, accumulate);

    // 막대그래프가 그려지는 영상을 생성한다.
    int hist_w = 512, hist_h = 400;
    int bin_w = cvRound((double)hist_w / histSize);     // 상자의 폭
    Mat histImage(hist_h, hist_w, CV_8UC3, Scalar(0, 0, 0));

    // 값들이 영상을 벗어나지 않도록 정규화한다.
    normalize(b_hist, b_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());
    normalize(g_hist, g_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());
    normalize(r_hist, r_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());
```

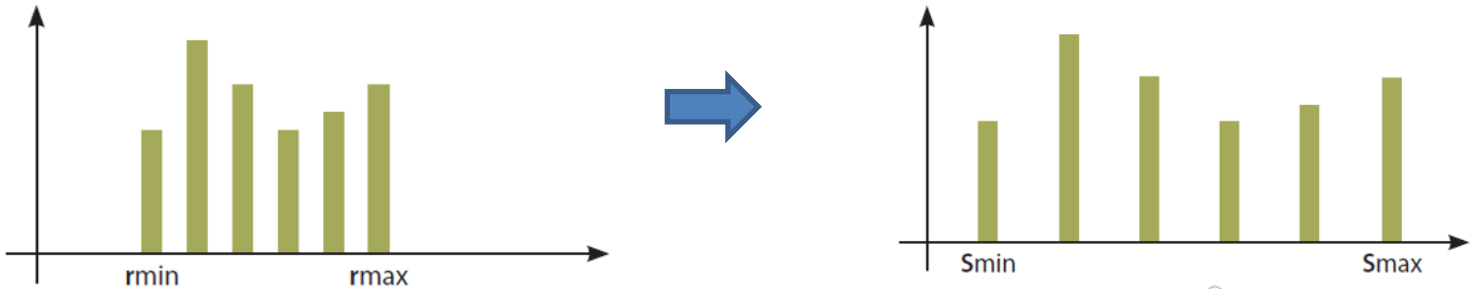
OpenCV 함수로 히스토그램 계산하기

```
// 히스토그램의 값을 막대로 그린다.
   for (int i = 0; i < 255; i++) {
      line(histImage, Point(bin_w*(i), hist_h), Point(bin_w*(i), hist_h - b_hist.at<float>(i)), Scalar(255, 0, 0));
      line(histImage, Point(bin_w*(i), hist_h), Point(bin_w*(i), hist_h - g_hist.at<float>(i)), Scalar(0, 255, 0));
      line(histImage, Point(bin_w*(i), hist_h), Point(bin_w*(i), hist_h - r_hist.at<float>(i)), Scalar(0, 0, 255));
   }

   imshow("입력 영상", src);
   imshow("컬러 히스토그램", histImage);
   waitKey();
   return 0;
}
```

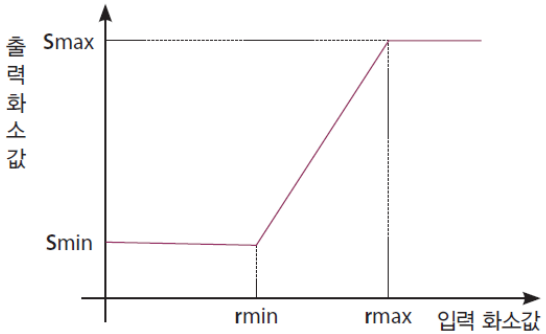


히스토그램 스트레칭

- 목적
  - 영상의 대비 (contrast) 증가하여 가시도 향상



- 계산식



$$s = \frac{(s_{\max} - s_{\min})}{(r_{\max} - r_{\min})} \times (r - r_{\min}) + s_{\min}$$

히스토그램 스트레칭

- 계산예
  - 원영상 히스토그램

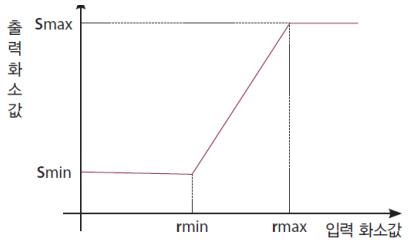
화소값	0	1	2	3	4	5	6
화소개수	0	0	30	60	20	0	0

  - 스트레칭

$r_{\min}$=2, $r_{\max}$=4, $s_{\min}$=0, $s_{\max}$=6

$$s = \frac{(s_{\max} - s_{\min})}{(r_{\max} - r_{\min})} \times (r - r_{\min}) + s_{\min}$$

r	s
2	0
3	3
4	6



화소값	0	1	2	3	4	5	6
화소개수	30	0	0	60	0	0	20

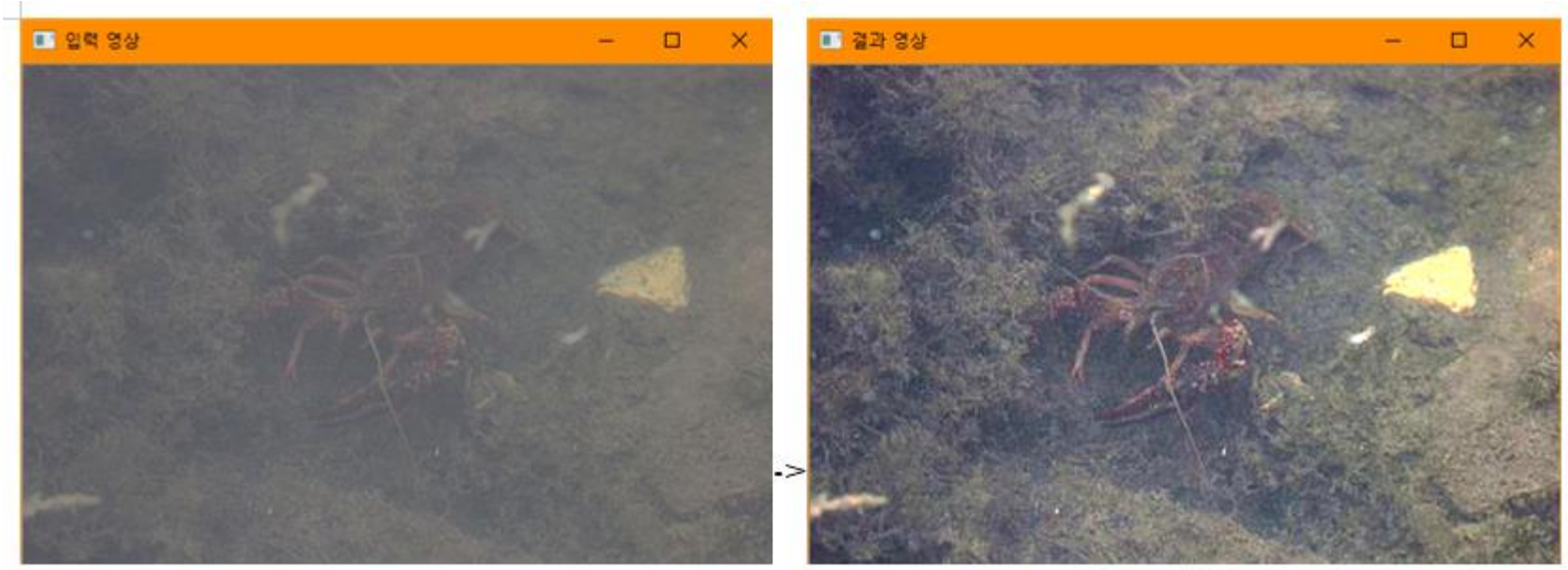
히스토그램 스트레칭

```
int stretch(int x, int r1, int s1, int r2, int s2)
{
    float result;
    if (0 <= x && x <= r1) {
        result = s1 / r1 * x;
    }
    else if (r1 < x && x <= r2) {
        result = ((s2 - s1) / (r2 - r1)) * (x - r1) + s1;
    }
    else if (r2 < x && x <= 255) {
        result = ((255 - s2) / (255 - r2)) * (x - r2) + s2;
    }
    return (int)result;
}
```

```
int main()
{
    Mat image = imread("d:/crayfish.jpg");
    Mat new_image = image.clone();

    int r1, s1, r2, s2;
    cout << "r1를 입력하시오: " ; cin >> r1;
    cout << "r2를 입력하시오: "; cin >> r2;
    cout << "s1를 입력하시오: " ; cin >> s1;
    cout << "s2를 입력하시오: " ; cin >> s2;

    for (int y = 0; y < image.rows; y++) {
        for (int x = 0; x < image.cols; x++) {
            for (int c = 0; c < 3; c++) {
                int output = stretch(image.at<Vec3b>(y, x)[c], r1, s1, r2, s2);
                new_image.at<Vec3b>(y, x)[c] =
                                        saturate_cast<uchar>(output);
            }
        }
    }
    imshow("입력영상", image);
    imshow("출력영상", new_image);
    waitKey();
    return 0;
}
```

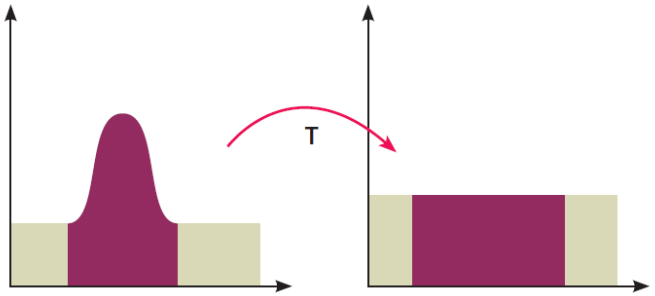
히스토그램 스트레칭

- 실행결과

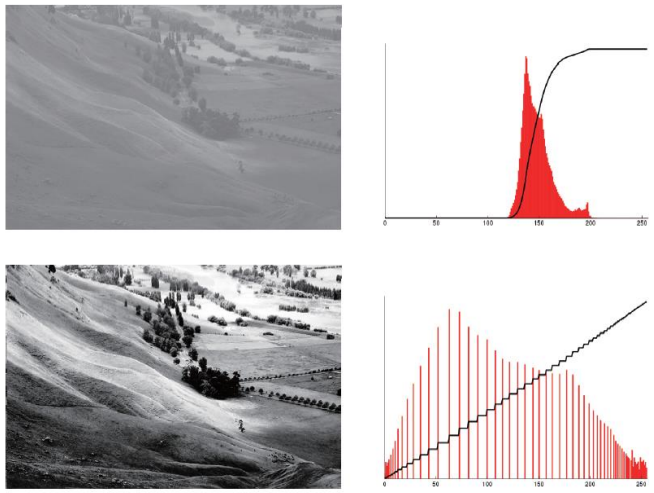


히스토그램 평활화

- 목적
  - 히스토그램이 균일하게 되도록 변환하는 처리



  - 영상 밝기 분포의 폭을 넓혀서 가시도 향상



히스토그램 평활화

- 방법

  (step1) 원 영상에 대한 히스토그램 $H(i)$ 을 구한다

  (step2) 누적 히스토그램 $H'(i)$을 구한다.

  (step3) 화소값 i 에 대한 새로운 화소값 s 를 계산한다.

$$s = G_{max}\frac{H\prime(i)}{n_t}$$

G_{max}: 최대 밝기값, n_t: 총 화소 수

30	36	42	30
25	30	33	37
34	46	25	50
40	37	32	28

$H(i)$

명도값	25	28	30	32	33	34	36	37	40	42	46	50
개수	2	1	3	1	1	1	1	2	1	1	1	1

$H'(i)$

명도값(i)	25	28	30	32	33	34	36	37	40	42	46	50
축적 히스토그램(H(i))	2	3	6	7	8	9	10	12	13	14	15	16

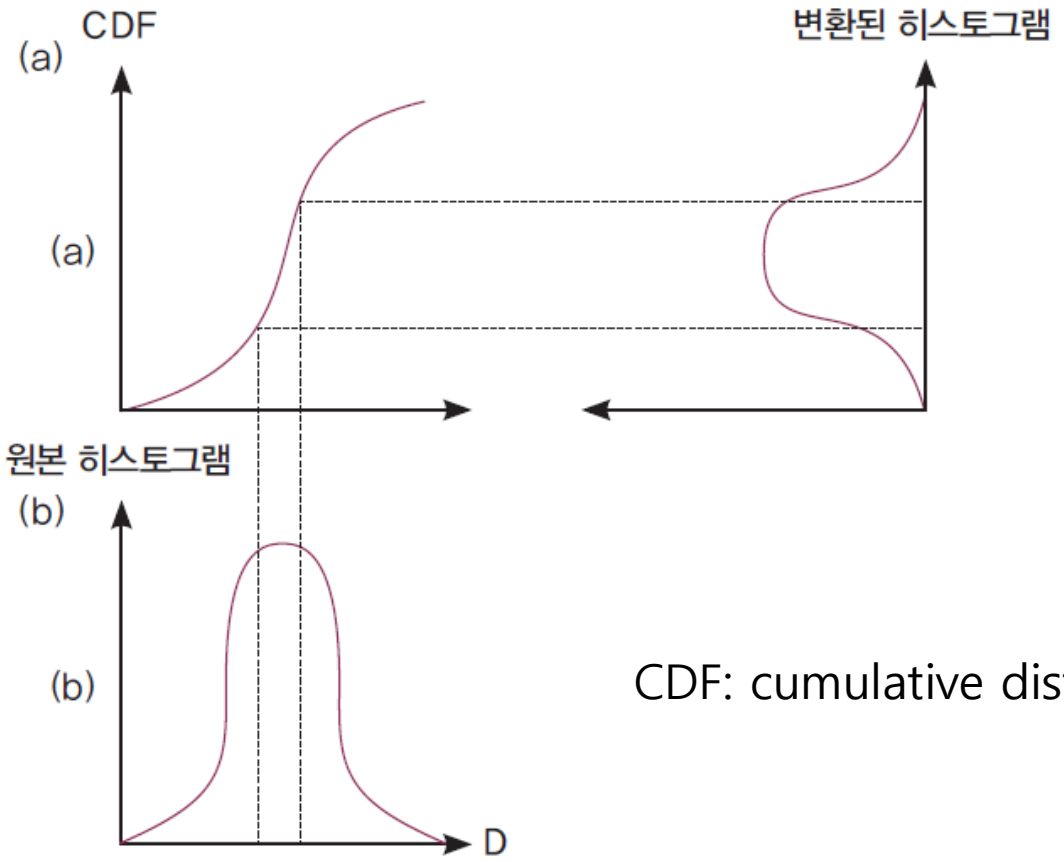
s

명도값(v)	25	28	30	32	33	34	36	37	40	42	46	50
축적 히스토그램(H(v))	2	3	6	7	8	9	10	12	13	14	15	16
새로운 명도값	32	48	96	112	127	143	159	191	207	223	239	255

96	159	223	96
32	96	127	191
143	239	32	255
207	191	112	48

$G_{max} = 255, n_t = 16$

히스토그램 평활화



CDF: cumulative distribution function

히스토그램 평활화

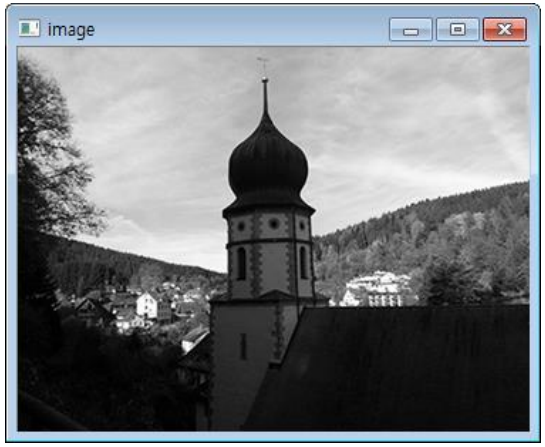
```
01  #include <opencv2/opencv.hpp>
02  using namespace cv;
03  using namespace std;
04
05  void calc_Histo(const Mat& image, Mat& hist, int bins,  int range_max = 256 ) { ... }
06
07  void draw_histo(Mat hist, Mat &hist_img, Size size = Size(256, 200)) {  ...  }
08
09  void create_hist(Mat img, Mat &hist, Mat &hist_img)
10  {
11      int  histsize = 256, range = 256;
12      calc_Histo(img, hist, histsize, range);          // 히스토그램 계산
13      draw_histo(hist, hist_img);                      // 히스토그램 그래프 그리기
14  }
15
16  int main()
17  {
18      Mat image = imread("../image/equalize_test.jpg", 0);        // 명암도 영상 읽기
19      CV_Assert(!image.empty());                                  // 영상파일 예외처리
20      Mat hist, dst1, dst2, hist_img, hist_img1, hist_img2;
21      create_hist(image, hist, hist_img);              // 히스토그램 및 그래프 그리기
22
```

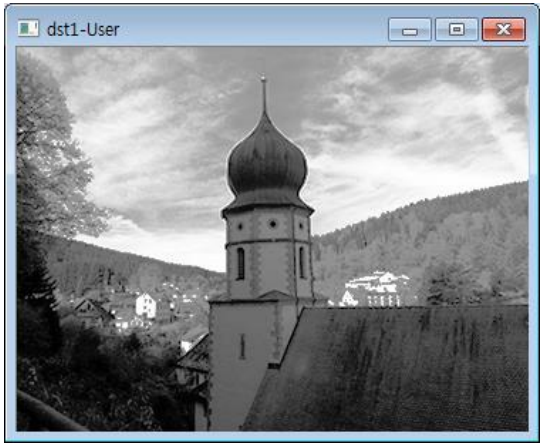
히스토그램 평활화

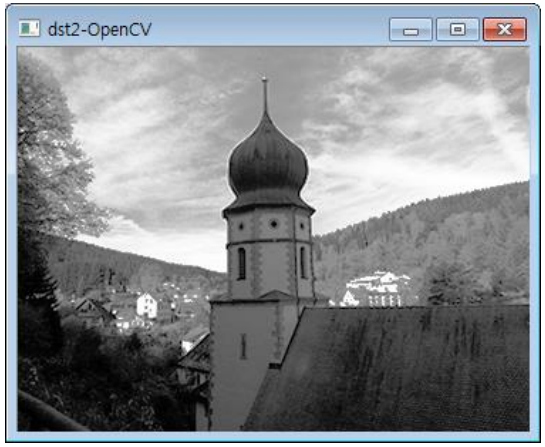
```
23      // 히스토그램 누적합 계산
24      Mat accum_hist = Mat(hist.size(), hist.type(), Scalar(0));
25      accum_hist.at<float>(0) = hist.at<float>(0);
26      for (int i = 1; i < hist.rows; i++){
27           accum_hist.at<float>(i) = accum_hist.at<float>(i - 1) + hist.at<float>(i);
28      }
29
30      accum_hist /= sum(hist)[0];                                  // 누적합의 정규화
31      accum_hist *= 255;
32      dst1= Mat(image.size(), CV_8U);
33      for (int i = 0; i < image.rows; i++) {
34           for (int j = 0; j < image.cols; j++) {
35                int idx = image.at<uchar>(i, j);
36                dst1.at<uchar>(i, j) = (uchar)accum_hist.at<float>(idx);
37           }
38      }
```

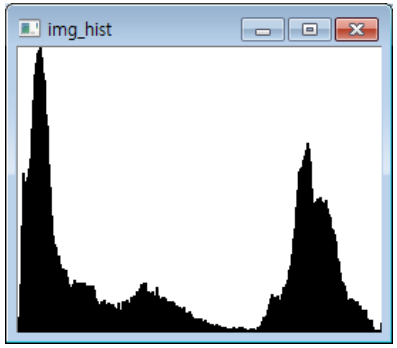
```
44      equalizeHist(image, dst2);                          // OpenCV 히스토그램 평활화
45      create_hist(dst1, hist, hist_img1);                 // 히스토그램 및 그래프 그리기
46      create_hist(dst2, hist, hist_img2);
47
48      imshow("image", image),      imshow("img_hist", hist_img);      // 원본 히스토그램
49      imshow("dst1-User", dst1),   imshow("User_hist", hist_img1);    // 사용자 평활화
50      imshow("dst2-OpenCV", dst2), imshow("OpenCV_hist", hist_img2); // OpenCV 평활화
51      waitKey();
52      return 0;
53  }
```

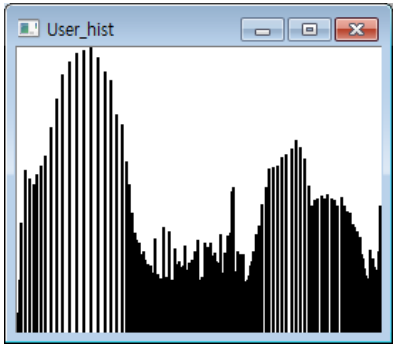
히스토그램 평활화

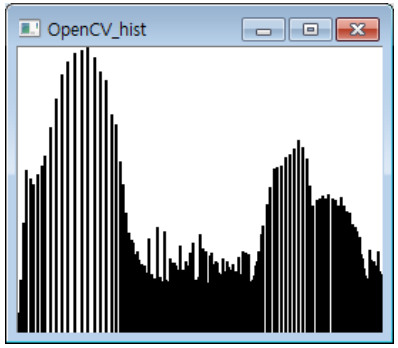






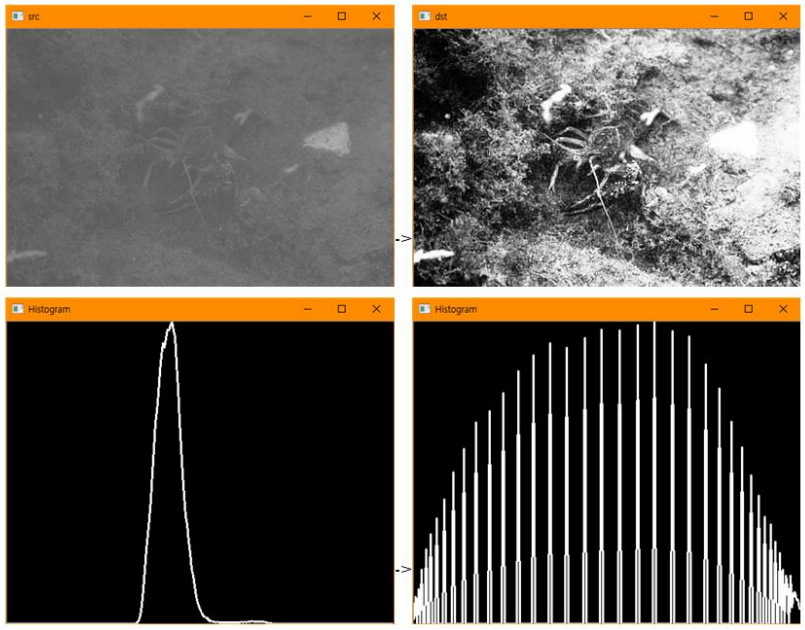




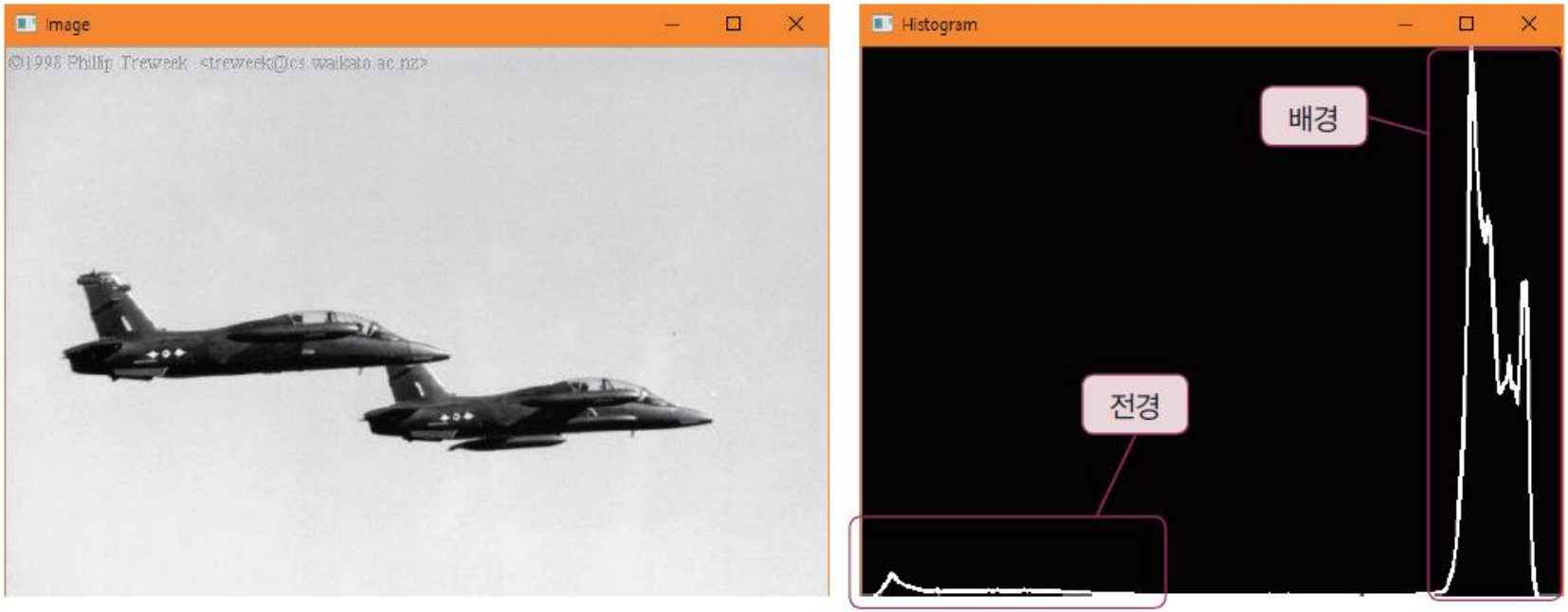


히스토그램 평활화

```
int main()
{
    Mat src = imread("d:/crayfish.jpg", IMREAD_GRAYSCALE);
    if (src.empty()) { return -1; }

    Mat dst;
    equalizeHist(src, dst);

    imshow("Image", src);
    imshow("Equalized", dst);
    waitKey(0);
    return 0;
}
```

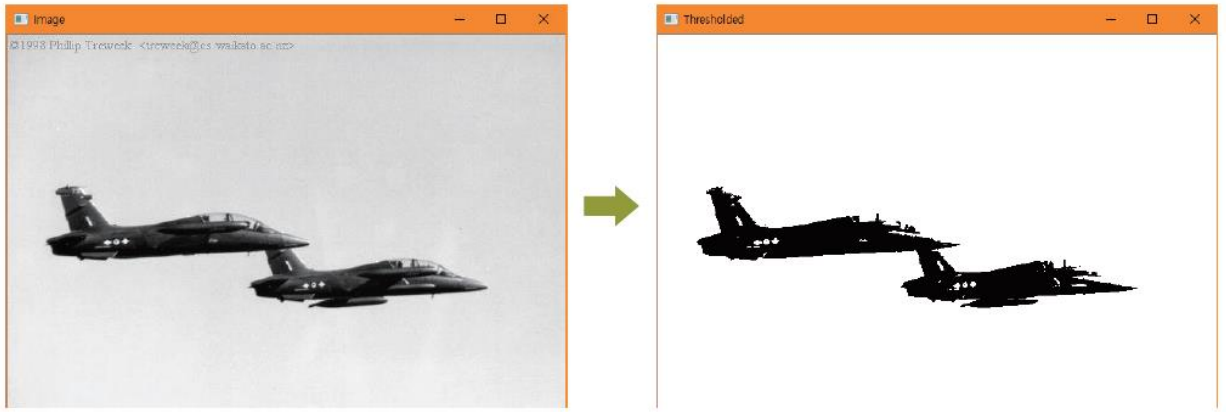


히스토그램을 이용한 전경과 배경 분리



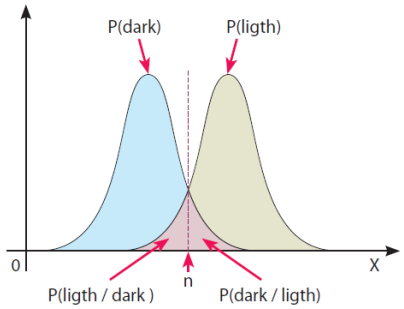
히스토그램을 이용한 전경과 배경 분리

```
using namespace std;
using namespace cv;

int main()
{
    Mat src, dst;

    src = imread("d:/plane.jpg", IMREAD_GRAYSCALE);
    imshow("Image", src);
    if (!src.data) { return -1; }

    Mat threshold_image;
    threshold(src, threshold_image, 100, 255, THRESH_BINARY);
    imshow("Thresholded", threshold_image);
    waitKey(0);
    return 0;
}
```

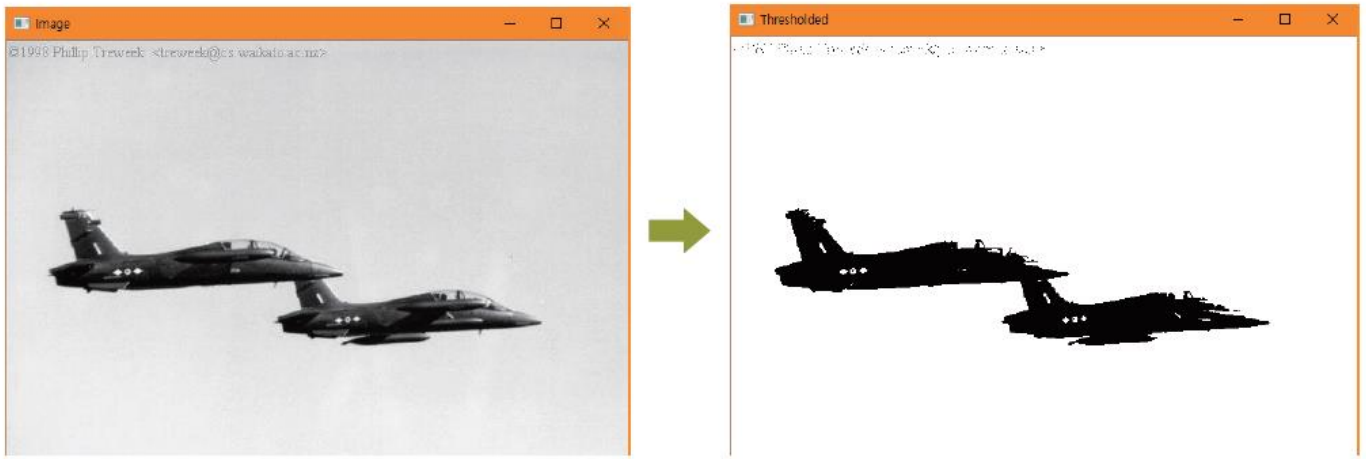


향상된 이진화 방법

- 컴퓨터가 영상을 분석하여 자동으로 임계값을 결정하게 할 수 없을까?



threshold(src, threshold_image, 0, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);



HW

1. 우측과 같은 히스토그램을 갖는 영상이 있을 때
   1) 히스토그램 스트레칭 후의 히스토그램을 구하라
   2) 히스토그램 평활화 후의 히스토그램을 구하라

화소값	화소수
0	0
1	0
2	50
3	60
4	50
5	20
6	10
7	0

2. 영상처리에서 투영 (projection) 은 다음 수식으로 표현된다.

$$histo\_v(x) = \sum_{y=0}^{h-1} f(x,y)$$

$$histo\_v(y) = \sum_{x=0}^{w-1} f(x,y)$$

, $f(x,y)$ = x,y좌표의 화소값

OpenCV 함수 중에서 cv::reduce() 함수를 이용해서 수직 및 수평방향 투영하는 프로그램을 작성하고, 영상 파일을 읽어서 투영 히스토그램을 출력하라.

3. 영상 파일을 읽어 윈도우에 표시하고, 마우스 이벤트를 통해서 드래그할 때 선택된 영역의 R, G, B 히스토그램을 세 개의 윈도우에 그리는 프로그램을 작성하라.