점처리 : 기본
(픽셀단위 처리)

점처리 (point processing)

- 영상처리의 가장 기본적인 처리 기법
- 영상데이터에 대한 사용자 접근의 시작
- 영상을 취급하는 디지털 전자기기에서 많이 사용함

- 하나하나의 단위픽셀 각각에 독립적으로 작용하는 처리

점처리 vs. 영역처리

20	20	20	20	20	20	20
50	50	60	60	60	60	60
50	50	60	70	70	70	60
60	60	60	70	70	80	70
60	70	70	70	80	80	70
70	80	70	80	80	90	80
80	80	80	80	90	90	90

– 70을 변화시키기 위해 (4,4)의 자신만의 데이터를 사용한 경우

vs.

– (4,4)위치 인근의 여러 점들을 사용하는 경우

상수값에 의한 산술연산

덧셈연산	$OutImg[x][y] = InImg[x][y] + C_1$
뺄셈연산	$OutImg[x][y] = InImg[x][y] - C_2$
곱셈연산	$OutImg[x][y] = InImg[x][y] * C_3$
나눗셈연산	$OutImg[x][y] = InImg[x][y] / C_4$

- 상수값 더하기 : 밝기(brightness) 증가
- 상수값 곱하기 : 대비(contrast)의 증가

입력 영상

상수 더하기(원영상+60)

상수곱하기(원영상*1.4)

상수 빼기 (원영상-60)

```
for (i=0; i<Height; i++)
  for (j=0; j<Width; j++)
     OutImg[i][j] = InImg[i][j] + c;
```

상수값에 의한 산술연산

영상값 클램핑(Clamping)

- InImg[x][y], OutImg[x][y] 둘 다는 0~255사이의 값
- 연산결과도 0~255사이의 값을 가져야 함
- 값을 더하거나 조작하여 결과값은 0~255사이의 범위를 벗어나는 경우 방지 필요

```
for (i=0; i<Height; i++)
  for (j=0; j<Width; j++)
     OutImg[i][j] = InImg[i][j] + c;
```

```
for (i=0; i<Height; i++)
  for (j=0; j<Width; j++){
     int result = InImg[i][j] + c;
     if (result > 255) OutImg[i][j] = 255;
     else if (result < 0) OutImg[i][j] = 0;
     else OutImg[i][j] = result;
  }
```

클램핑 하지 않은 경우 다음에 out에는 어떤 값이 들어갈까?

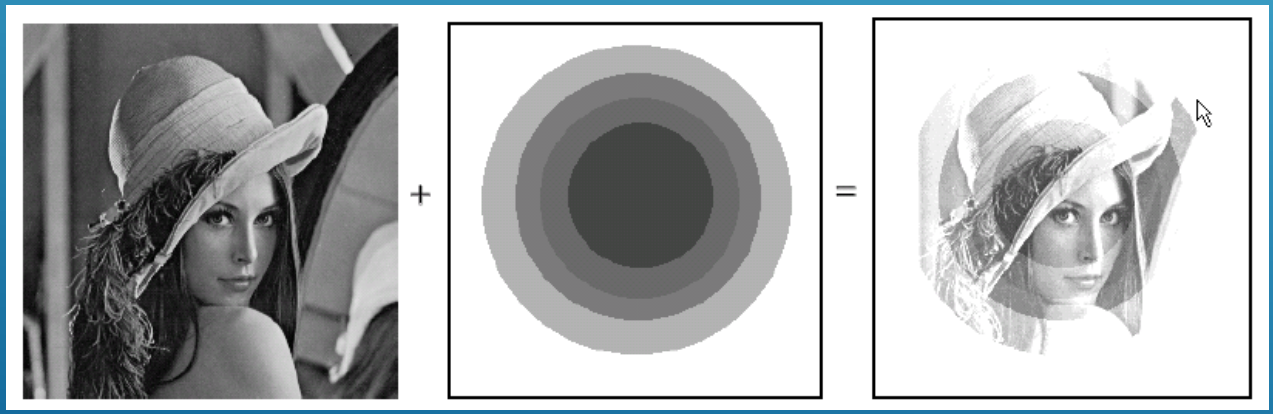
```
unsigned char   in=250;
unsigned char   out=in + 20;
```

두 영상 사이의 산술연산 (프레임 단위 연산)

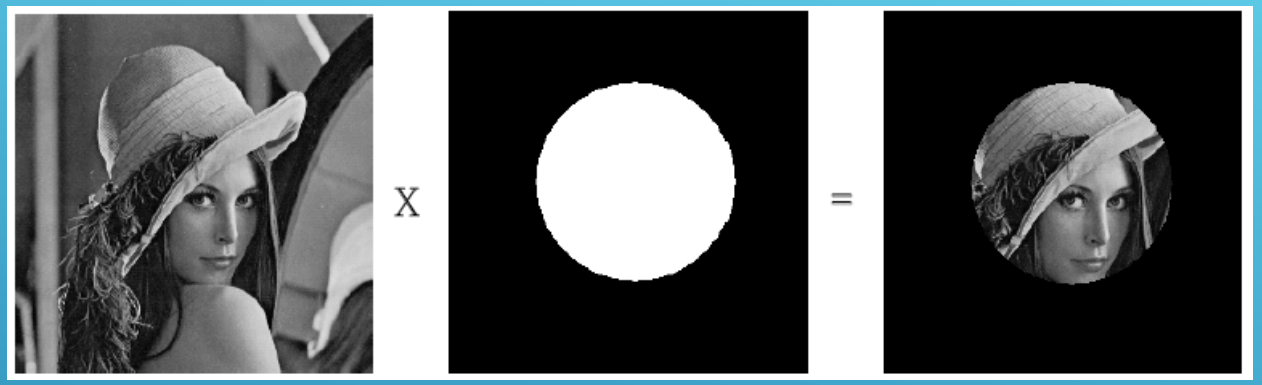
덧셈연산	OutImg[x][y] = InImg1[x][y] + InImg2[x][y]
뺄셈연산	OutImg[x][y] = InImg1[x][y] – InImg2[x][y]
곱셈연산	OutImg[x][y] = InImg1[x][y] * InImg2[x][y]
나눗셈연산	OutImg[x][y] = InImg1[x][y] / InImg2[x][y]

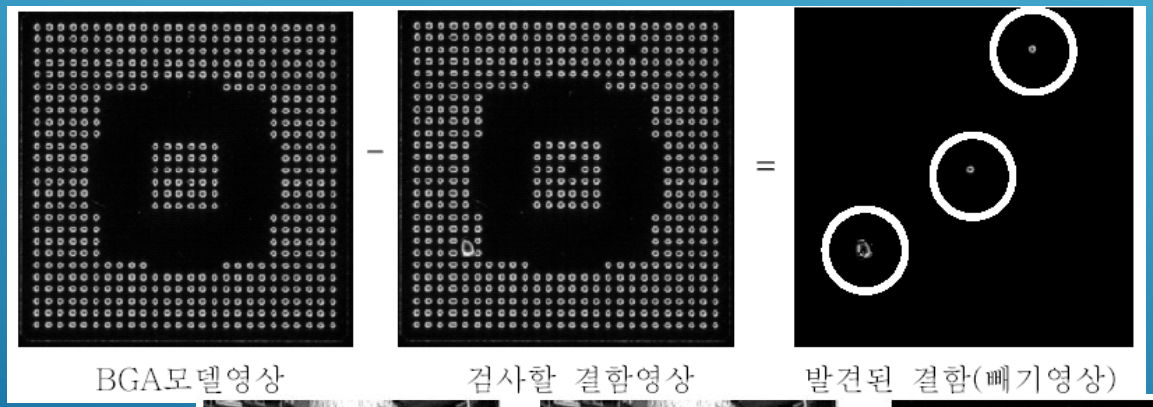
```
for (i=0; i<Height; i++)
  for (j=0; j<Width; j++){
     int result = InImg1[i][j] + InImg2[i][j];
     if (result > 255) OutImg[i][j] = 255;
     else if (result < 0) OutImg[i][j] = 0;
     else OutImg[i][j] = result;
  }
```

두 영상을 더할 때는 음수가 나올 수 없으므로 이 줄은 없어도 됨



두 영상 사이의 산술연산

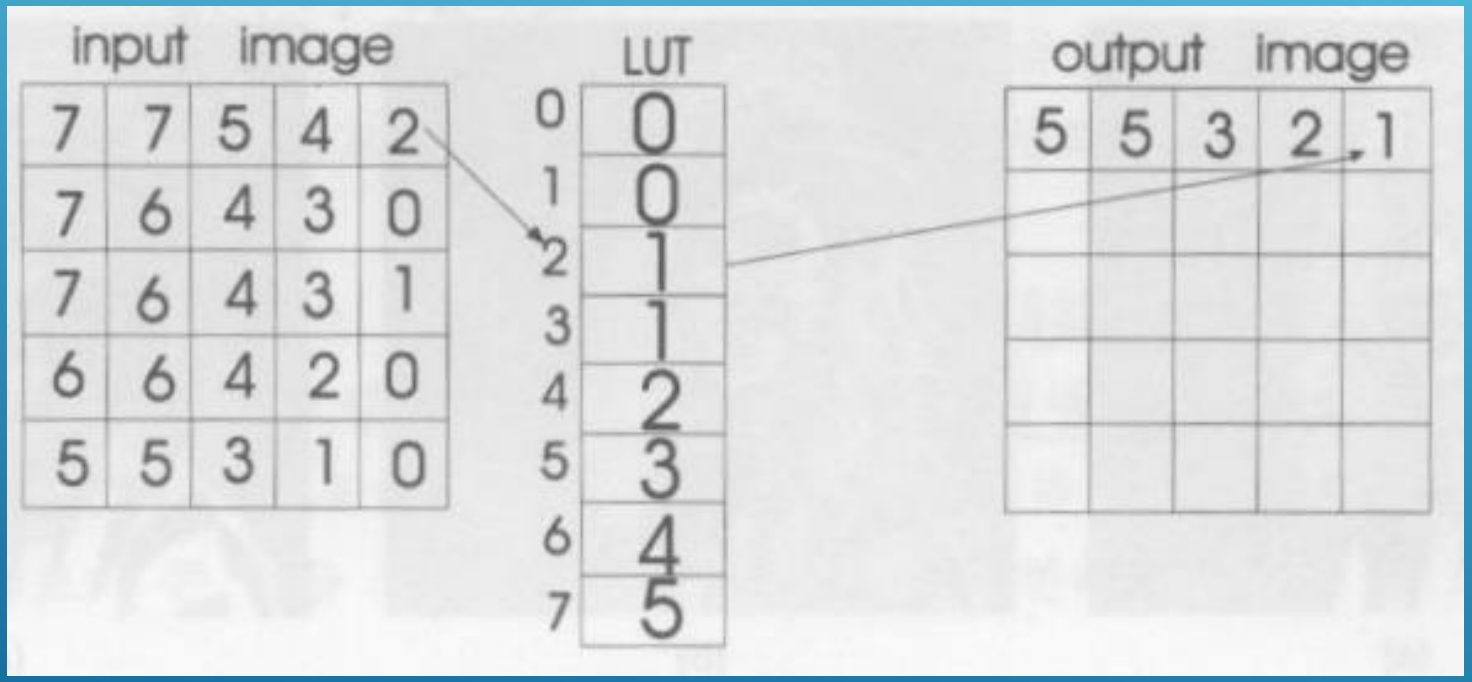




룩업 테이블 연산

룩업 테이블(Lookup Table: LUT) 연산

- 산술연산의 고속수행을 위해 사용
- LUT 배열을 미리 생성함에 의해 반복 산술연산의 회피가 가능함
- 점처리 기법에서 가장 효과적으로 사용됨



3비트 LUT 를 이용한 연산

룩업 테이블 연산

예) LUT에 의한 고속 연산: 상수 곱에 의한 대비처리의 고속화

```
for ( i=0; i<height; i++) {
   for ( j=0; j<width; j++)  {
      temp = (int)( InImg[i][j]*1.4); // 곱셈연산의 반복적 발생
      OutImg[i][j]= temp > 255 ? 255 : temp; // if 연산의 반복 발생
   }
} // 곱 연산, if연산의 횟수: height*width 번
```

```
unsigned char LUT[256]; // LUT로 사용할 메모리를 선언

// LUT값을 계산
for (i=0; i<256; i++) {
                int temp=(int)( i*1.4 );
                LUT[i] = temp > 255 ? 255 : temp;
}

// LUT를 통하여 영상을 처리
for (i=0; i<height; i++)
   for (j=0; j<width; j++)
      OutImg[i][j]=LUT[ InImg[i][j] ];
 // 곱 연산, if연산 횟수: 256번, 대입연산 width*height 번
```

룩업 테이블 연산

감마(γ) 상관관계에 의한 영상매크로 변환
- CRT 명암조절과 비슷한 비선형 변환
- 영상센서, 출력기 및 필름 등에서 사용함
- $0 < \gamma < 1$: 영상을 흐리게 만듬
- $\gamma > 1$: 영상을 밝게 만듬
- RGB모니터: $1.4 < \gamma < 2.8$

Output = Input $^{1/\gamma}$: Input, Output값이 0~1인 경우

$$Output = \left(\frac{Input}{255} \right)^{1/\gamma} * 255$$: Input, Output값이 0~255인 경우

```
#define GAMMA 2.0
double x;
for (i=0; i<255; i++) {
   x = pow((double)(x)/255.0, 1/GAMMA)*255.0;
   LUT[i] = (unsigned char) (x + 0.5);
}
for (i=0; i<height; i++)
   for (j=0; j<width; j++)
      outImg[i][j] = LUT[ inImg[i][j] ];
```

룩업 테이블 연산

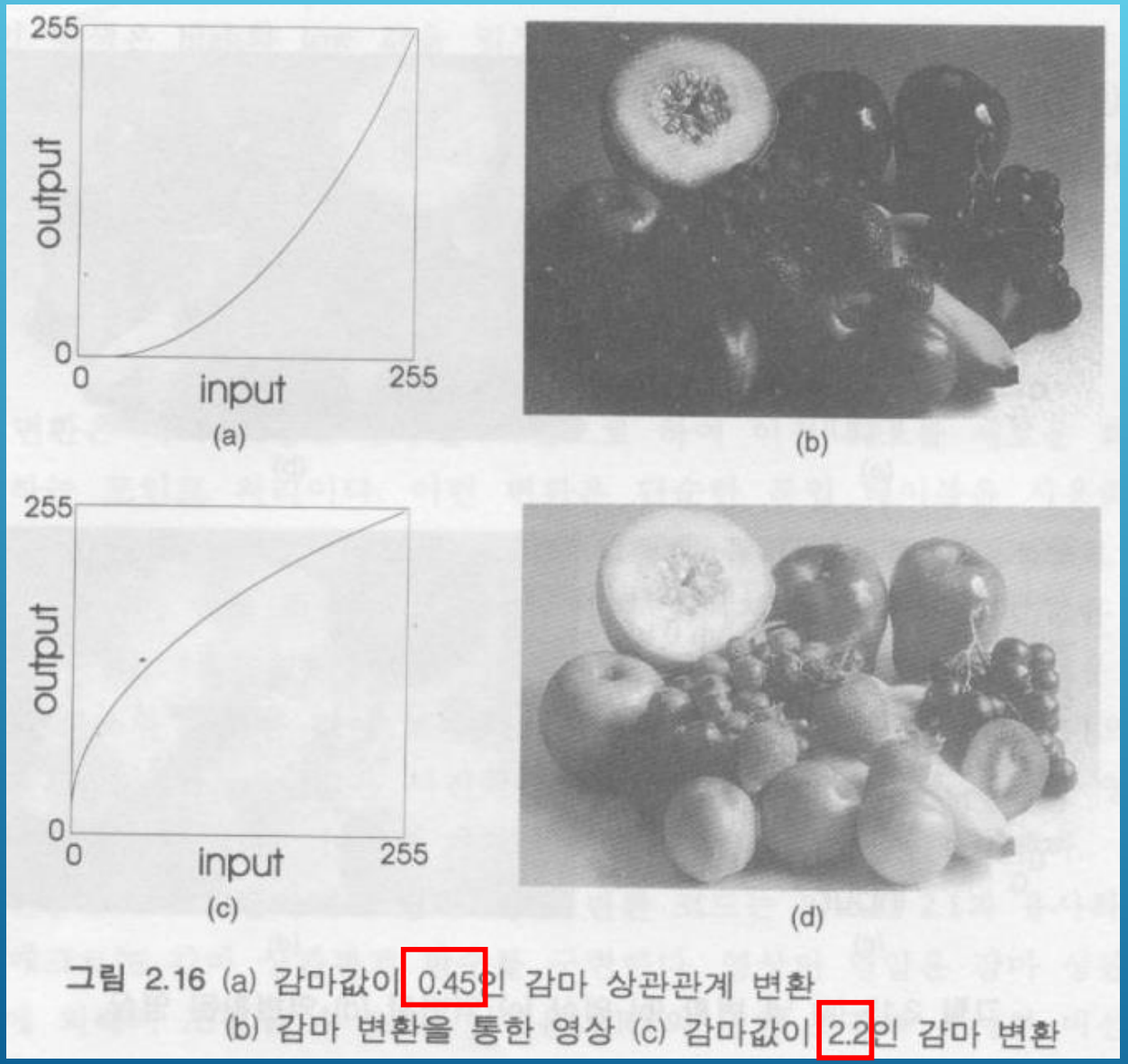


그림 2.16 (a) 감마값이 0.45인 감마 상관관계 변환
(b) 감마 변환을 통한 영상 (c) 감마값이 2.2인 감마 변환

-인간 시각은 똑같은 밝기 변화에 대해 어두운 부분의 변화에 더 민감하게 반응함

-그림 (d)는 어두운 부분에서의 밝기를 더 크게 바꾸어 주었으므로 가시화가 더 좋아지게 느낌

룩업 테이블 연산

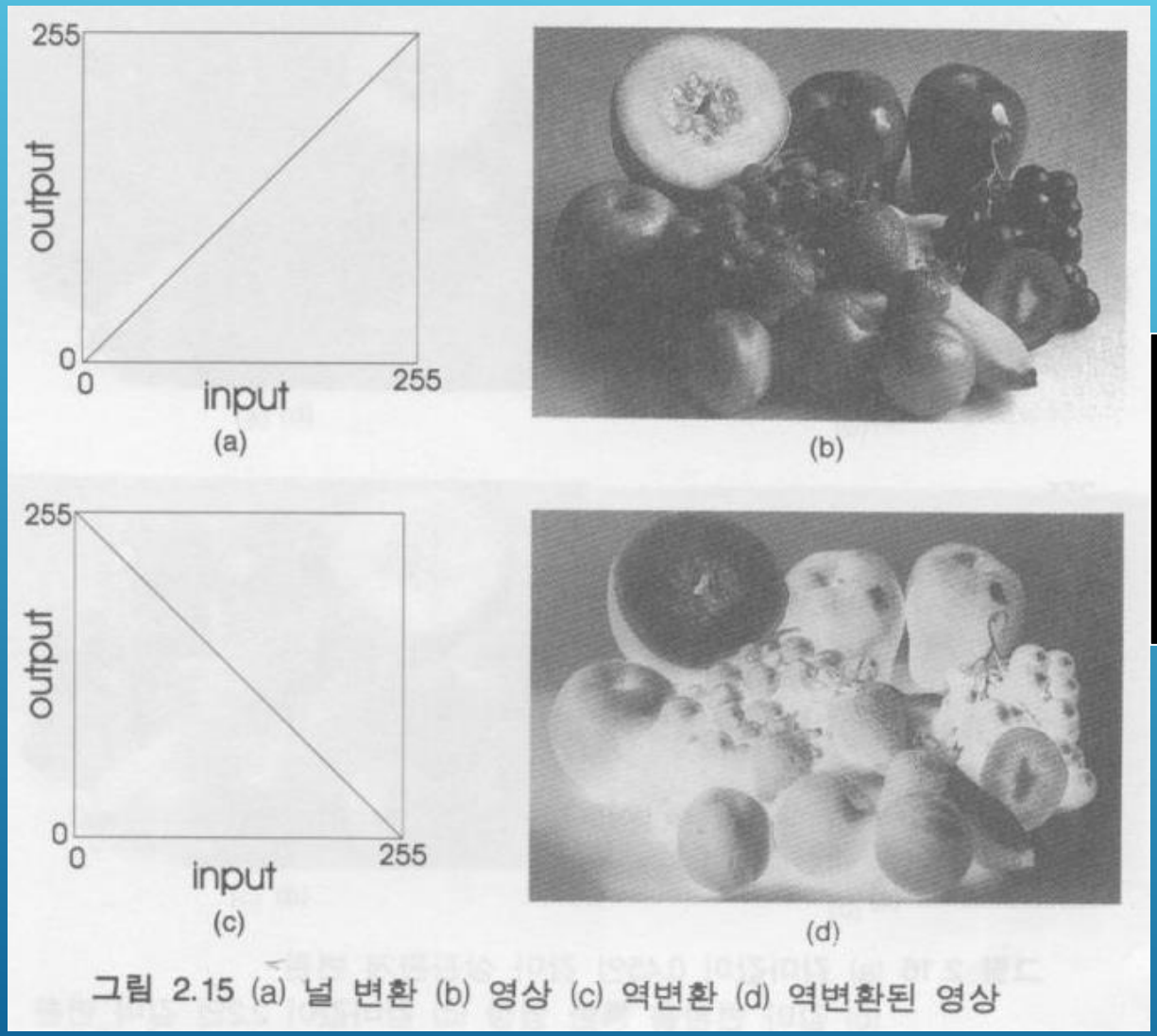
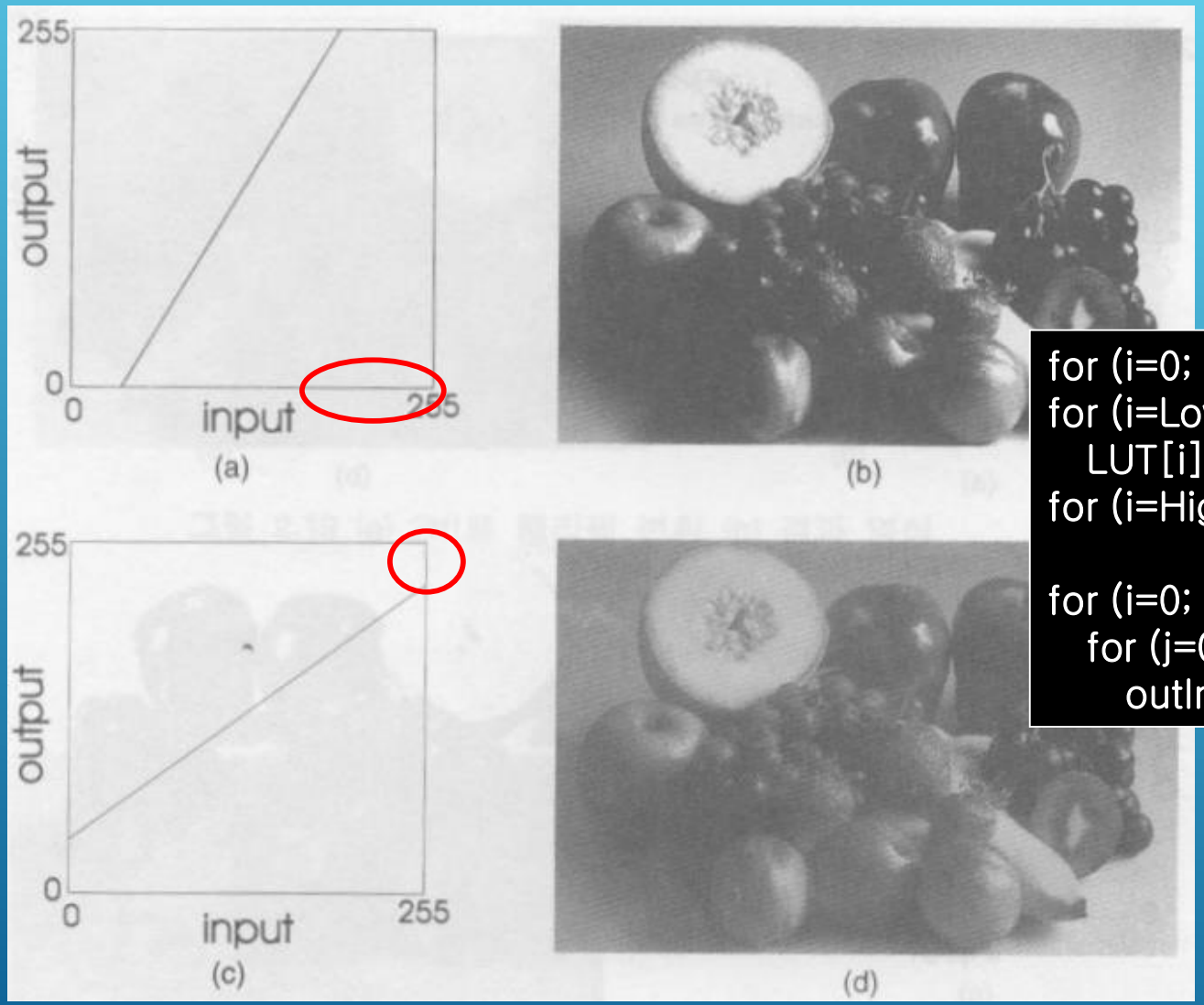


그림 2.15 (a) 널 변환 (b) 영상 (c) 역변환 (d) 역변환된 영상

```
for (i=0; i<255; i++)
   LUT[i] = 255-i;

for (i=0; i<height; i++)
   for (j=0; j<width; j++)
      outImg[i][j] = LUT[ inImg[i][j] ];
```

룩업 테이블 연산

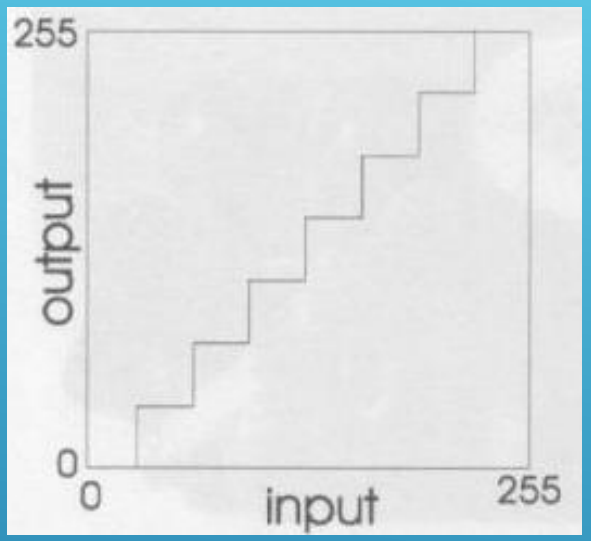


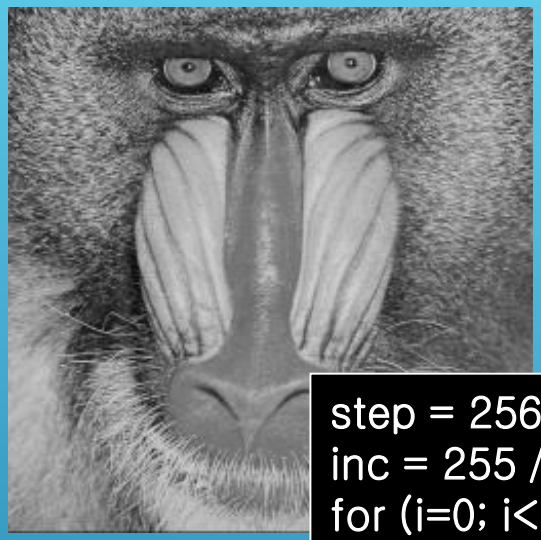
-명암대비변환:
명암이 스트레칭된 영상

-명암 대비 압축변환:
명암대비가 압축된 영상

```
for (i=0; i<Low; i++)  LUT[i] = 0;
for (i=Low; i<=High; i++)
   LUT[i] = (i-Low)/(High-Low)*255.0;
for (i=High+1; i<256; i++)  LUT[i] = 255;

for (i=0; i<height; i++)
   for (j=0; j<width; j++)
      outImg[i][j] = LUT[ inImg[i][j] ];
```

룩업 테이블 연산

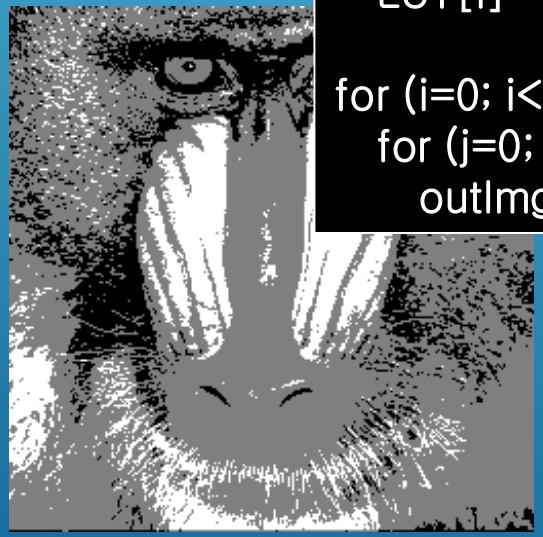




```
step = 256 / n_level;
inc = 255 / (n_level – 1);
for (i=0; i<255; i++)
   LUT[i] =(int) (i / step) * inc;

for (i=0; i<height; i++)
   for (j=0; j<width; j++)
      outImg[i][j] = LUT[ inImg[i][j] ];
```

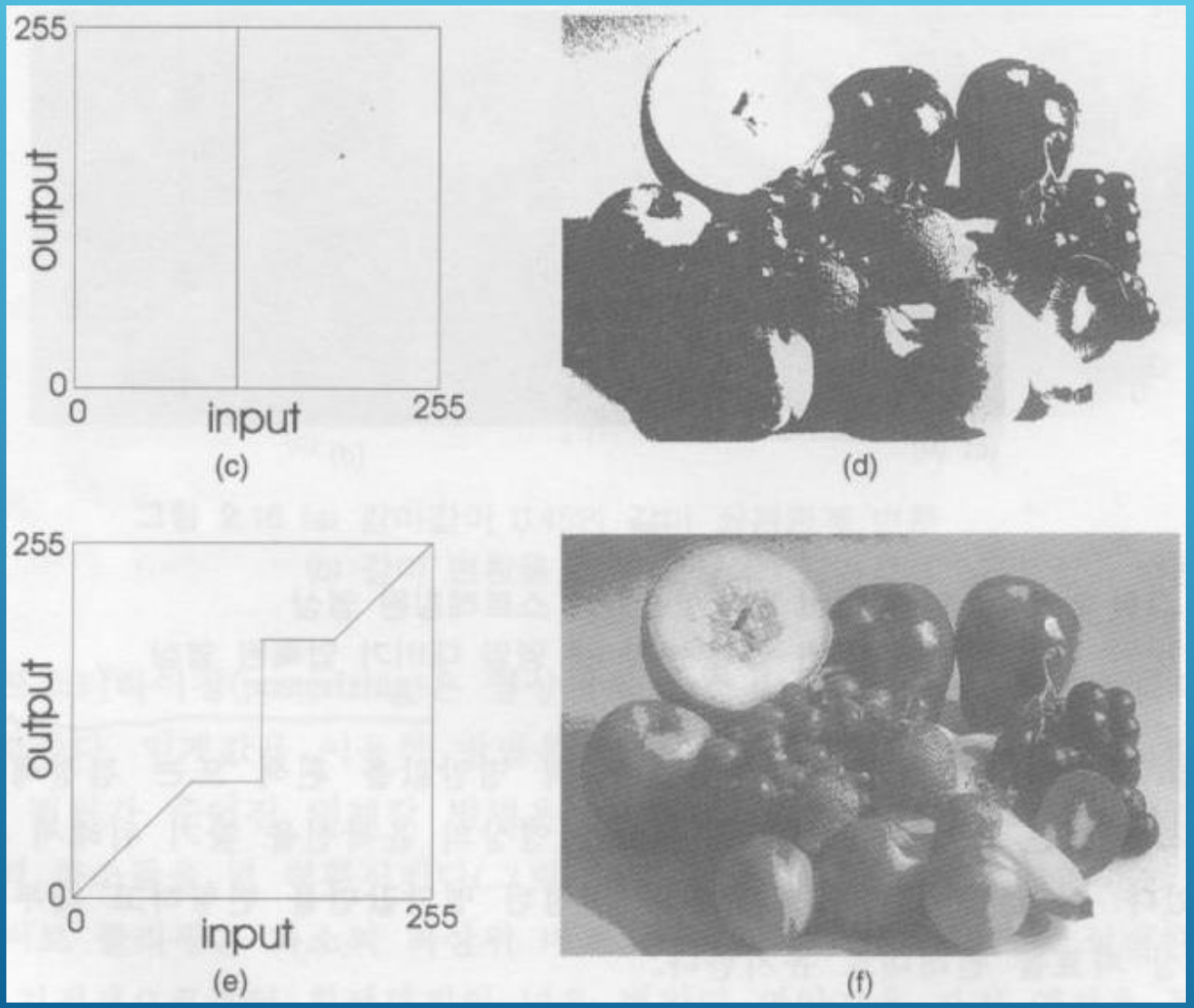
n_level == 3일 때 확인

8 레벨 포스트라이징(posterizing) 변환



3 레벨

룩업 테이블 연산



- 임계값 이용 변환:
  영상 이진화

- 범위가 주어지는 임계값 변환:
  특정범위 내에서만 이치화

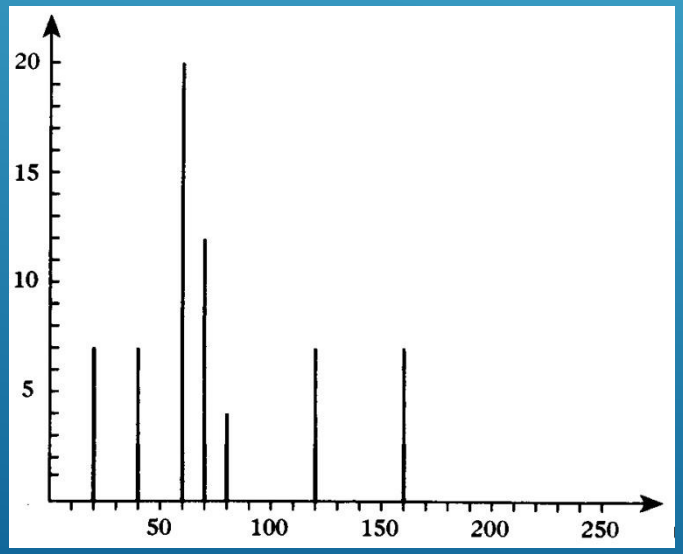
점처리 : 히스토그램 이용

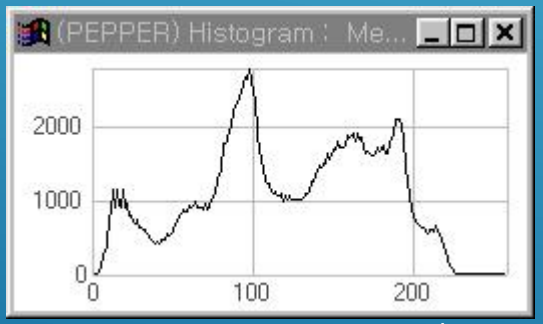
히스토그램(Histogram)?

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

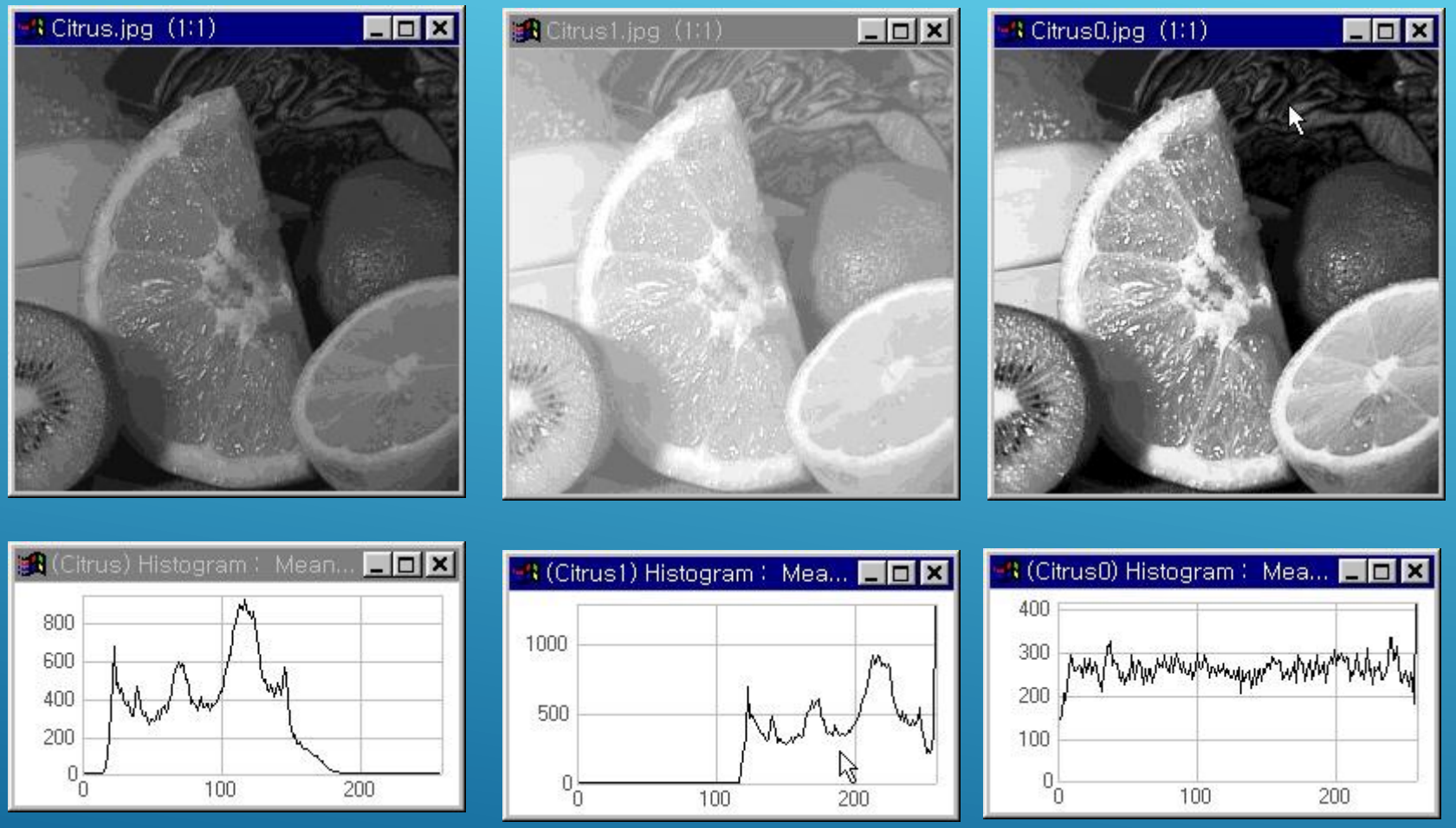
- 수평축: 영상의 밝기(intensity) 값
- 수직축: 수평축의 밝기값에 대응되는 크기를 가진 픽셀 수가 영상 안에서 몇 개나 있는지 나타내는 빈도수(frequency)

- 흑백영상의 경우 수평축은 0~255의 범위, 수직축의 값은 영상의 크기와 밝기의 분포에 따라 달라짐



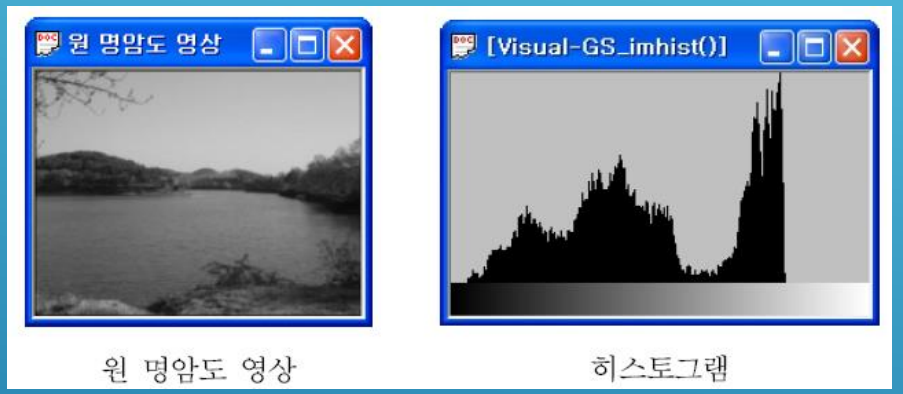


밝기분포가 다른 영상의 예



영상의 밝기 변화에 따른 히스토그램 차이

- **밝기 조절 결과**
  - 원 영상의 히스토그램과 비교하여 단지 20만큼 이동

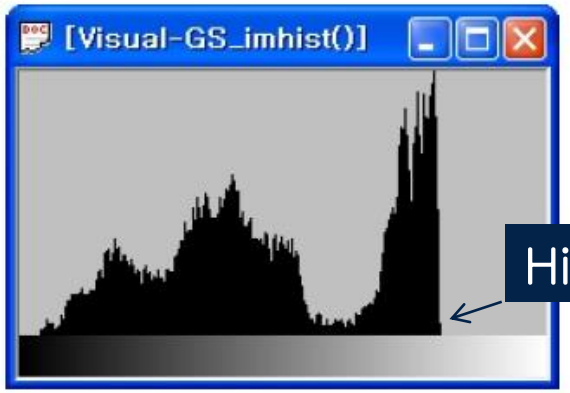


원 명암도 영상 히스토그램

밝기 조절 결과 명암도 영상 밝기 조절의 결과 히스토그램

- 명암대비 조절 결과
  - 분포 모양은 유지되나 확대/축소된 형태

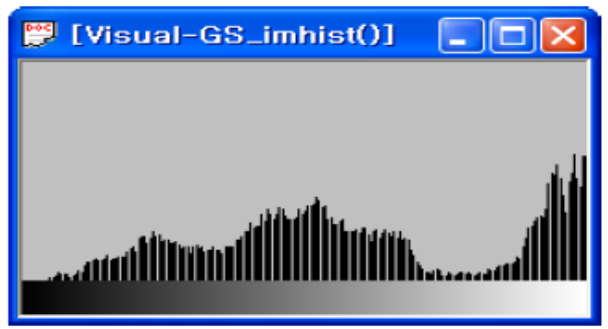
원 명암도 영상



히스토그램

OutImg[x][y]=InImg[x][y]*255/High

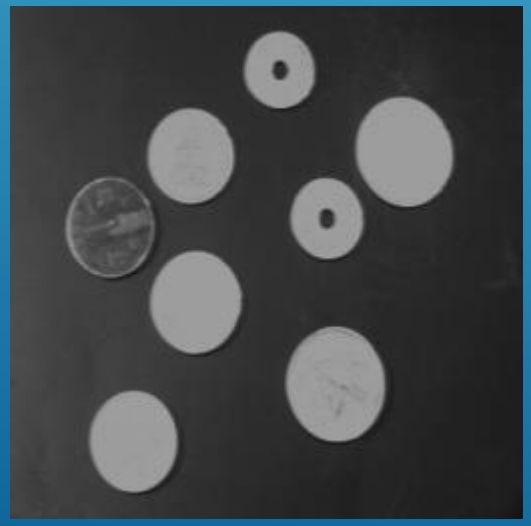
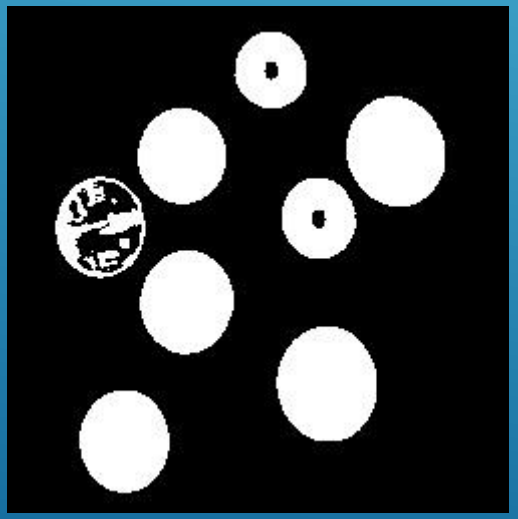
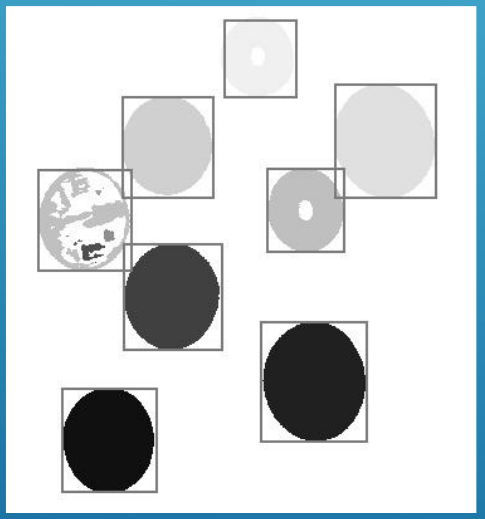
명암 대비 조절 결과 명암도 영상



명암 대비 조절 결과 히스토그램

히스토그램 용도


화질 향상

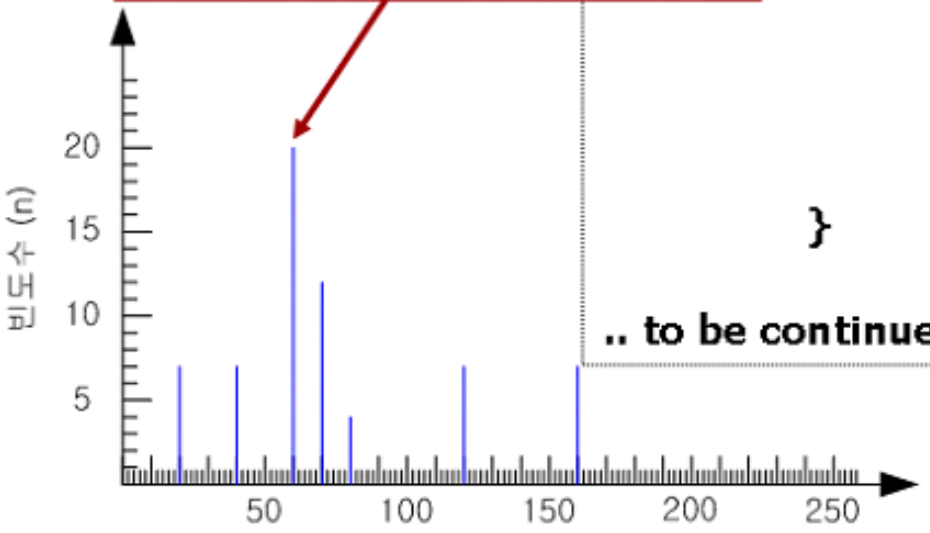




물체 인식 (이치화 후, 255값을 가지는 영역 및 위치를 자동추출)

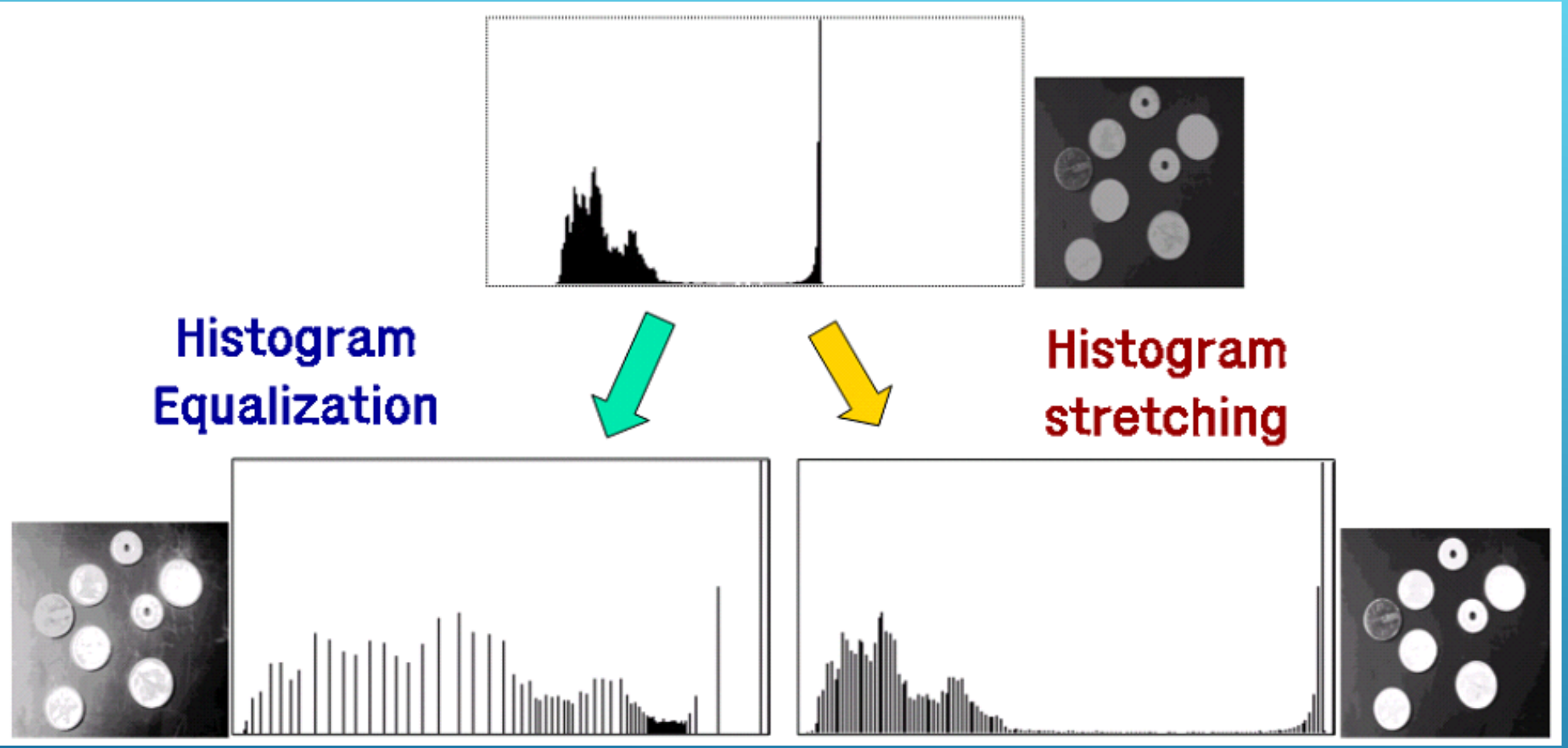
히스토그램 계산

```
void CWinTestDoc::m_ImgHisto(int height, int width)
{
        int i,j,vmax,vmin;
        for(i=0; i<256; i++) m_HistoArr[i]=0;

        for(i=0; i<height; i++)
        {
                for(j=0; j<width; j++)
                {
                        int gv = (int)m_InImg[i][j];
                        m_HistoArr[gv]++;
                }
        }
```

.. to be continued ..



히스토그램 평활화(equalization)



픽셀들이 여러 밝기 영역으로
넓게 퍼지며 분포하게 됨

양쪽으로 일정하게 당긴 것에 불과

둘다 히스토그램의 모양이 유지되는가?

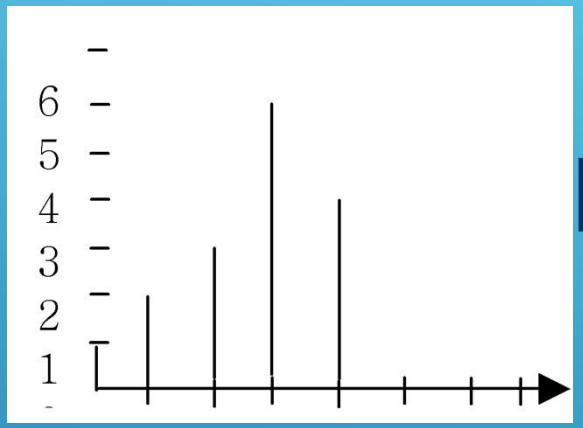
히스토그램 평활화 단계

c.f) 누적 히스토그램 H(i)는
항상 255근방에서 전체 픽셀수 N_t
와 같아지기 때문에 $h(255)=G_{max}$가 됨.
따라서, 평활화 식은 원본 히스토그램
을 0~255범위로 펴는 효과를 나타냄

평활화

- 누적히스토그램으로부터 **정규화합** 히스토그램을 구함

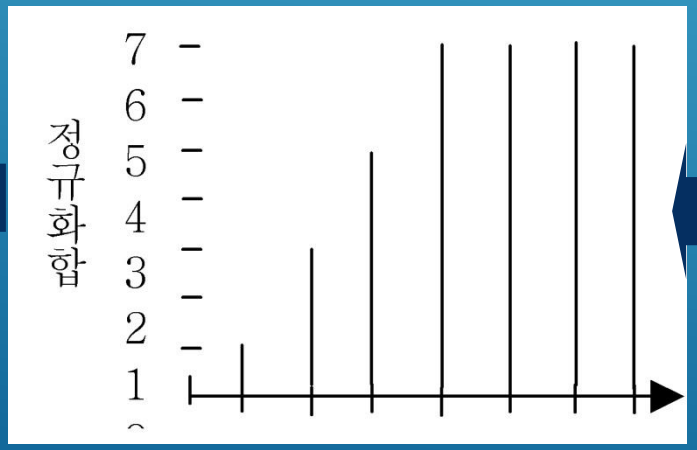
3	4	3	4
4	3	3	2
4	2	1	3
1	0	2	3



H(i): 누적 히스토그램
h(i): 정규화합 히스토그램

$$h(i) = \frac{G_{max}}{N_t} H(i)$$

인덱스	히스토그램	누적합	정규화합
0	1	1	0.44
1	2	3	1.31
2	3	6	2.62
3	6	12	5.25
4	4	16	7.0
5	0	16	7.0
6	0	16	7.0
7	0	16	7.0

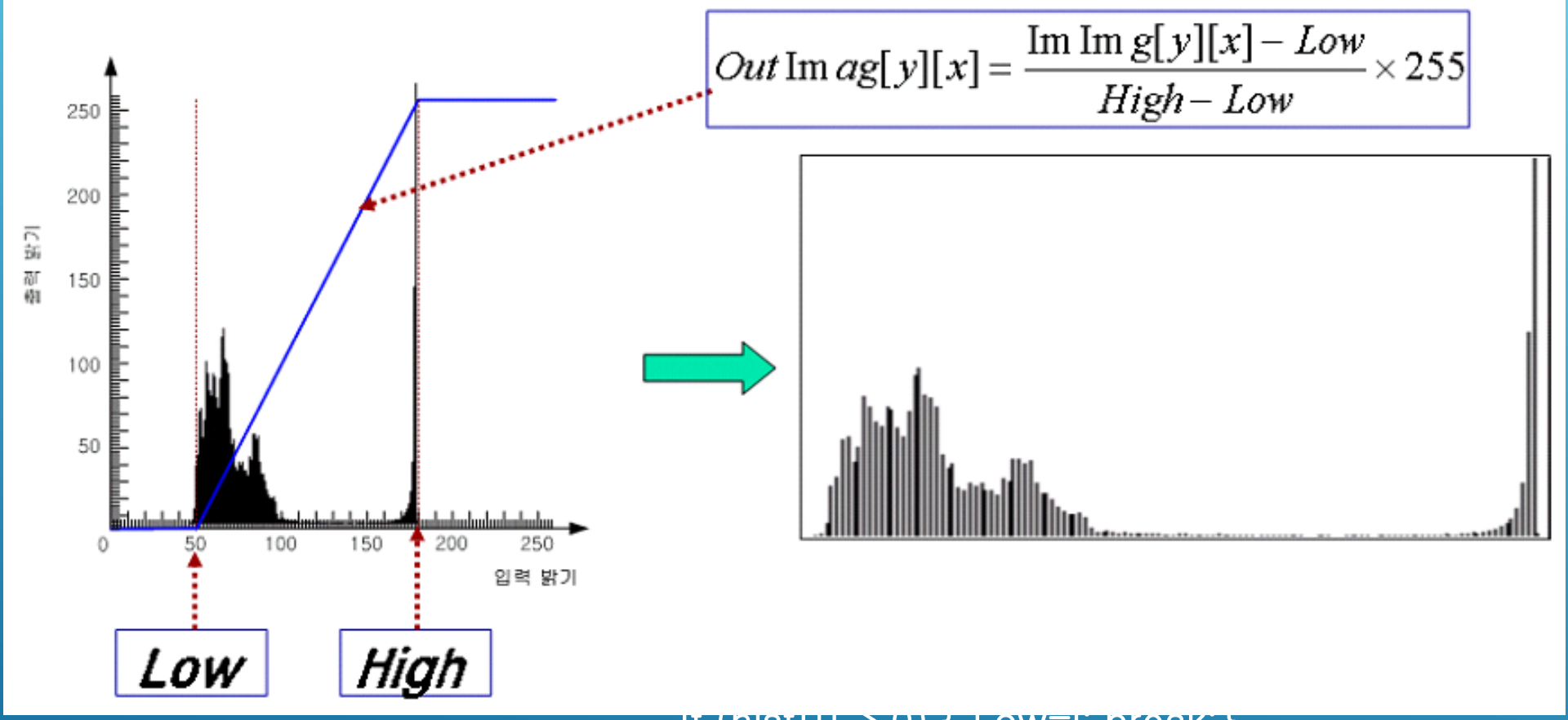


5	7	5	7
7	5	5	3
7	3	1	5
1	0	3	5

히스토그램 평활화 연산

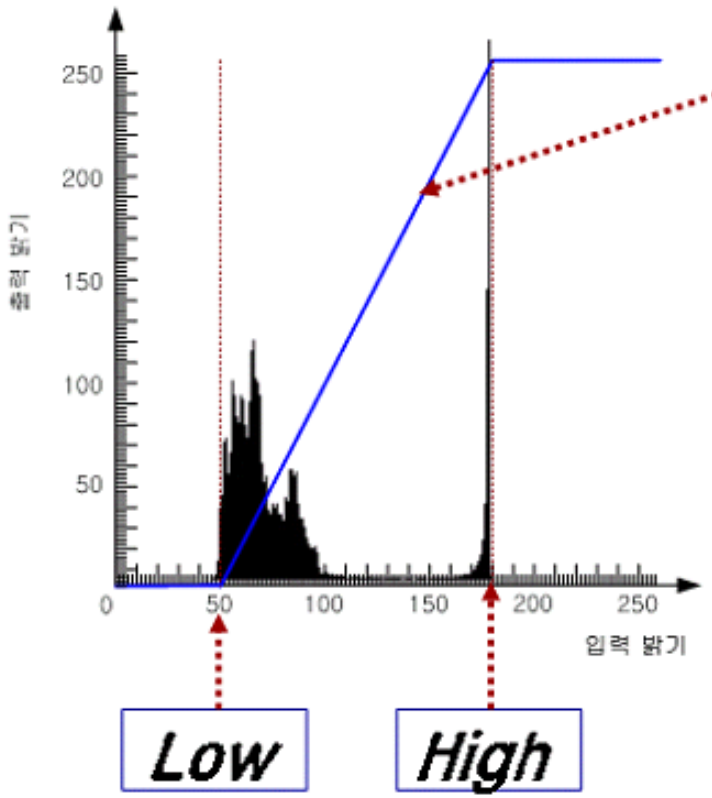
```
void m_HistoEqual(int height, int width)
{
        unsigned int *histogram = new unsigned int [256];
        unsigned int *sum_hist = new unsigned int [256];

        // histogram배열을 초기화
        for(i=0; i<256; i++) histogram[i]=0;

        // 영상의 histogram을 계산
        for(i=0; i<height; i++) for(j=0; j<width; j++) histogram[ m_InImg[i][j] ]++;

        int sum=0;
        float scale_factor=255.0f/(float)(height*width);
        // histogram의 정규화된 합 히스토그램을 계산
        for(i=0; i<256; i++)
        {
                sum += histogram[i];
                sum_hist[i] =(int)((sum*scale_factor) + 0.5);
        }

        // LUT로써 정규화합(sum_hist)배열을 사용하여 영상을 변환
        for(i=0; i<height; i++) // (변환영상은 m_OutImg에 저장)
                for(j=0; j<width; j++) m_OutImg[i][j]=sum_hist[m_InImg[i][j]];

        // 메모리를 해제
        delete []histogram; delete []sum_hist;
}
```

히스토그램 스트레칭(stretching)




```
    if (hist[i] > 0) { Low=i; break;}

for (i=255; i>0; i--)
    if (hist[i] > 0) {High=i; break;}
```

개선된 히스토그램 스트레칭 : ends-in search



$$Out\,\mathrm{Im}\,ag[y][x] = \frac{\mathrm{Im}\,\mathrm{Im}\,g[y][x] - Low}{High - Low} \times 255$$

앞에서처럼 High를 구하면 한 화소라도 250이면 다른 모든 화소가 180보다 작아도 High=250이 된다.

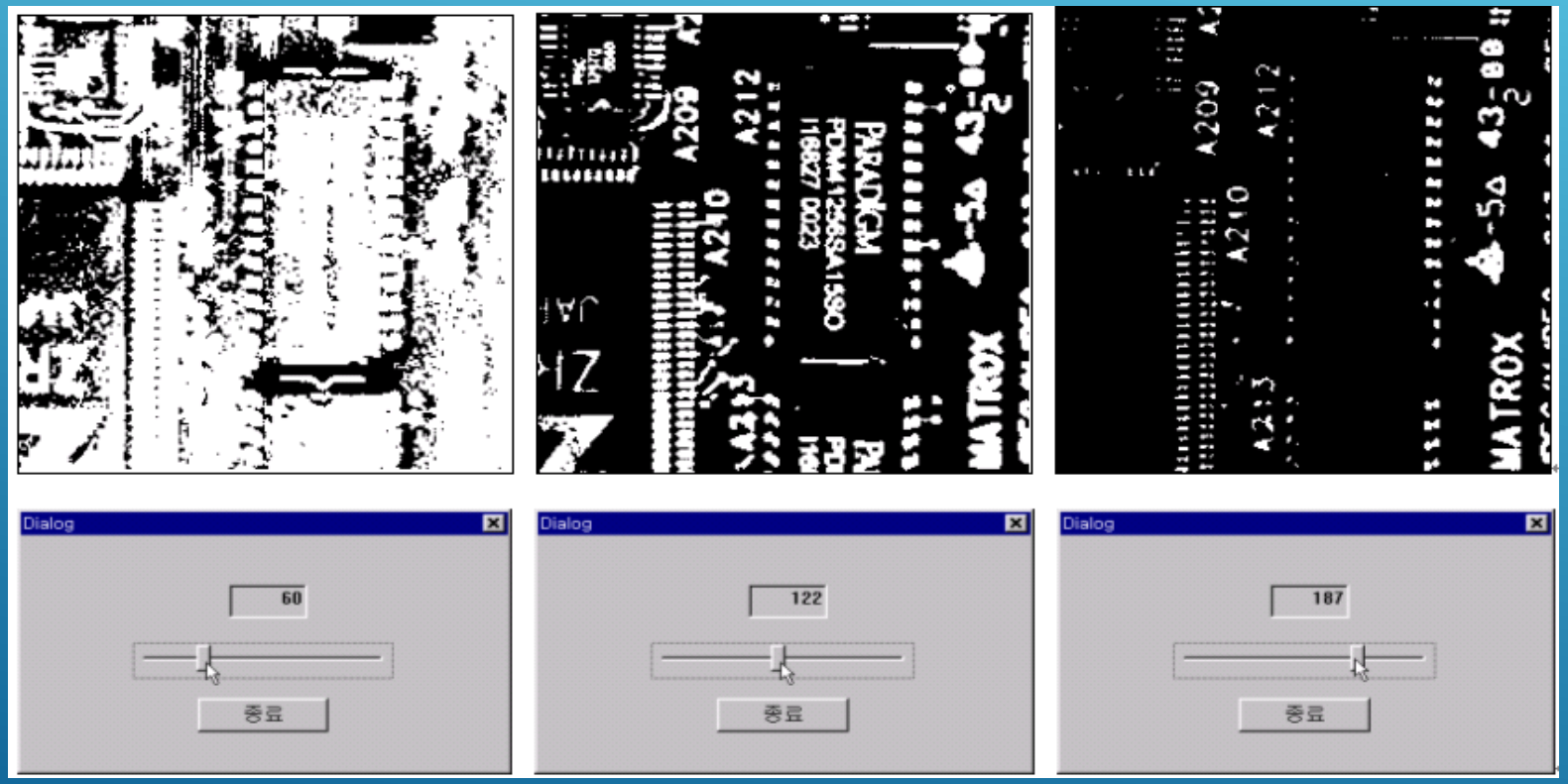
➔ 잡음에 매우 민감하다.

➔ 이를 줄이려면 p%개수만큼의 화소는 잡음으로 생각해서 Highr 계산할 때 버린다. (예를들어, 전체 화소개수가 300만개, p=0.01이면 300만*0.01%=300개는 버림.)

```
nr = width*height*p/100;
sum = 0;
for (i=255; i>0; i--) {
   sum += hist[i];
   if (sum > nr) {High=i; break;}  }
sum = 0;
for (i=0; i<255; i++) {
   sum += hist[i];
   if (sum > nr) {Low=i; break; }    }
```

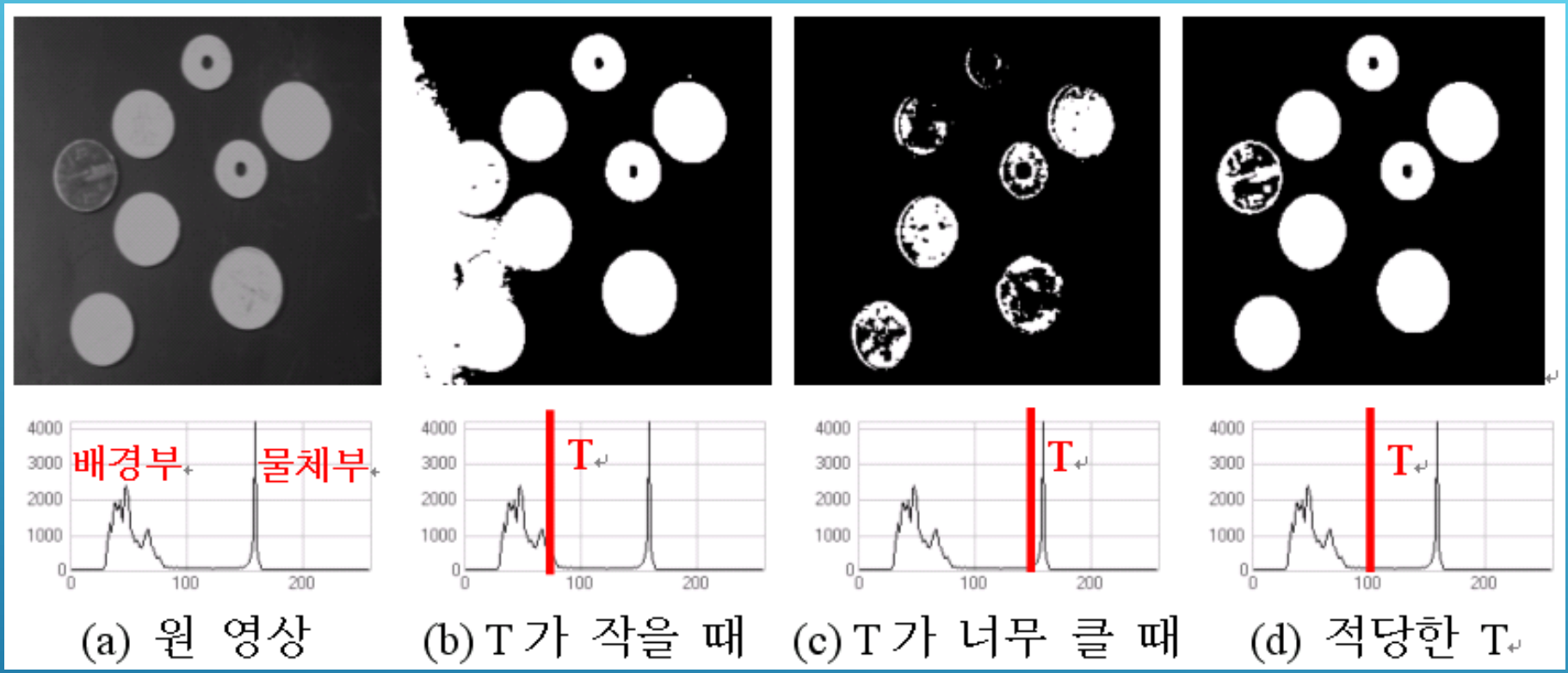
영상 이진화

$$OutImg\ [x][y] = \begin{cases} 0, & if\ InImg\ [x][y] \le T \\ 255, & otherwise \end{cases}$$



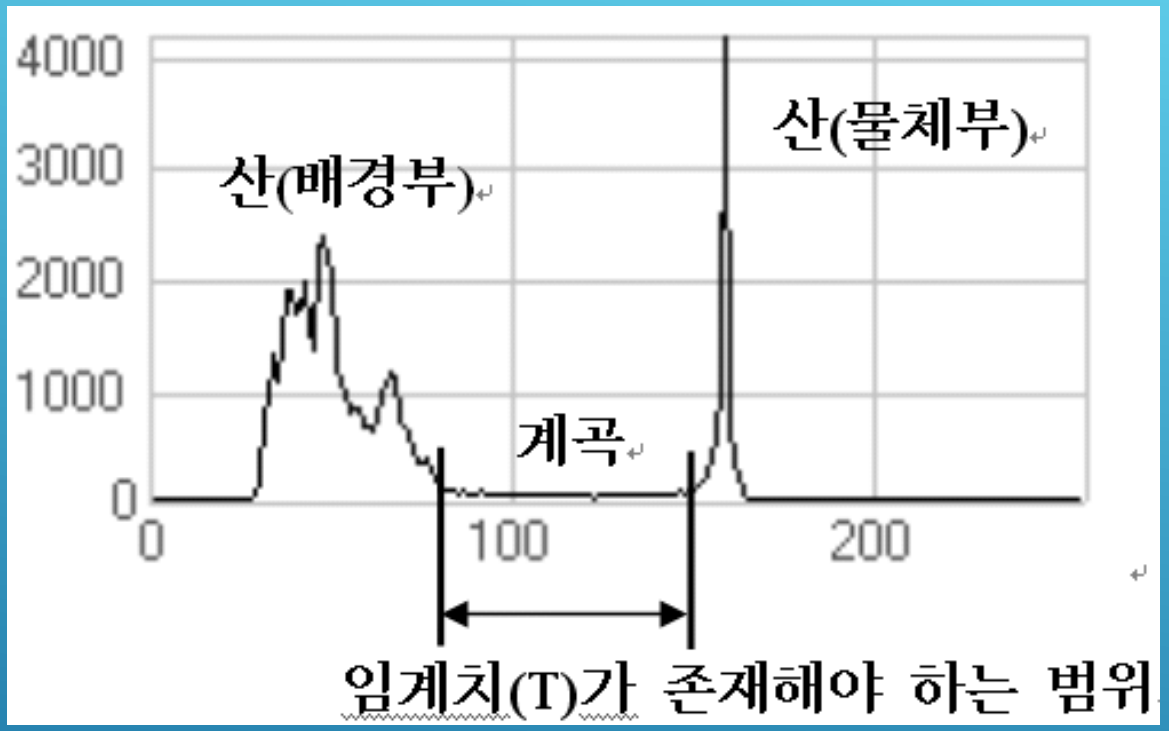
낮은 임계치 중간 임계치 높은 임계치

영상 이진화(binarization)



- 원본 영상: 픽셀의 밝기값이 0~255사이에 골고루 존재
- 이치화 영상: 픽셀의 밝기값이 0 아니면 255의 두 값(binary values) 중 하나를 가짐

이진화의 임계치(threshold)

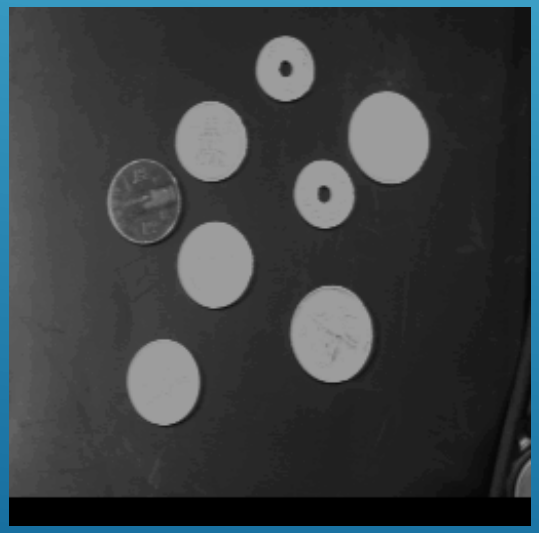


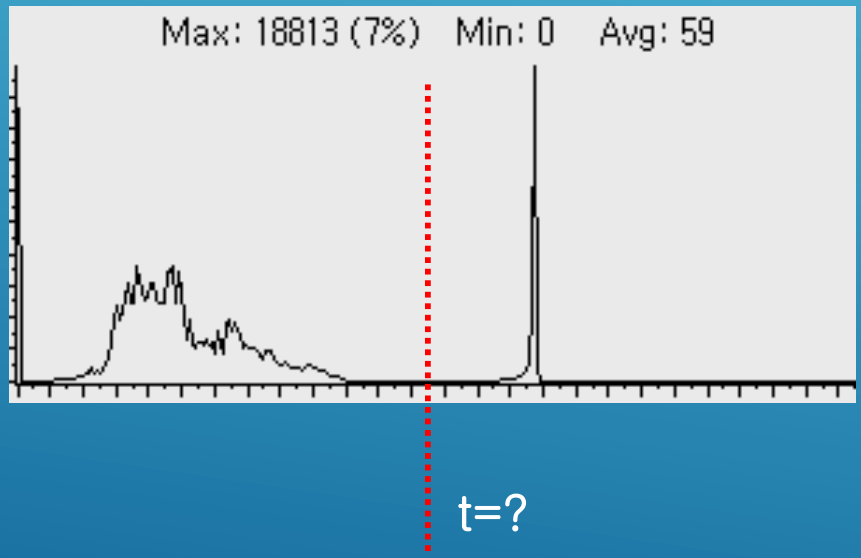
임계치는 사용자 인터페이스를 통해 제공하는 경우도 있고
자동으로 설정하도록 프로그램을 작성하는 경우도 있다.

자동 이진화 : Otsu 방법

Otzu Method

- Gray 영상에서 물체 및 배경 분리에 사용
- 히스토그램 작성 후, 히스토그램의 두 피크(peak)를 찾아냄
- 히스토그램 분포의 통계적 특성을 이용한 자동 임계값 결정
- 히스토그램 분포를 두 개의 정규분포로 모델링
- 평균과 t를 잘 선택해야 두 그룹의 분산값을 줄일 수 있음





•참고문헌: Computer and Robot Vision, R.M.Haralick, L.G. Shapiro, Addison Wesley

자동 이진화 : Otsu 방법

- 밝기값의 높은 유사성을 가진 픽셀그룹들의 분리
- 높은 유사성(high homogeneity)은 작은 분포값 (Low Variance)을 가짐
- 배경부와 물체부의 두 그룹이 가능한 낮은 분포값을 가지도록 임계치 t를 정해줌
- 구현: t를 0~255로 바꾸면서 σ_W^2 값이 가장 작은 t를 찾음

$$\sigma_W^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

$P(i) = \#\{(r,c) \mid \text{Image}(r,c) == i\} / (\# R \times C)$

: 밝기 i의 히스토그램 확률

$q_1(t) = \sum_{i=1}^{t} P(i)$: t보다 작은 픽셀그룹 확률

$q_2(t) = \sum_{i=t+1}^{I} P(i)$: t보다 큰 픽셀그룹 확률

$$\mu_1(t) = \sum_{i=1}^{t} iP(i)/q_1(t) \qquad \sigma_1(t) = \sum_{i=1}^{t} [i-\mu_1(t)]^2 P(i)/q_1(t)$$

$$\mu_2(t) = \sum_{i=t+1}^{I} iP(i)/q_2(t) \qquad \sigma_2(t) = \sum_{i=t+1}^{I} [i-\mu_2(t)]^2 P(i)/q_2(t)$$

c.f) 일반평균과 확률평균

. 일반 평균은 u = (Σ i)/n 이며, 이 때는 각 요소에 대한 weighting이 1 이고, 값을 더해준 후, 개수 n을 나누어 줌

. 확률평균의 경우 u = Σ i * p(i) 이고, 개수를 나누어줄 필요 없음. Σp(i) =1 이기 때문

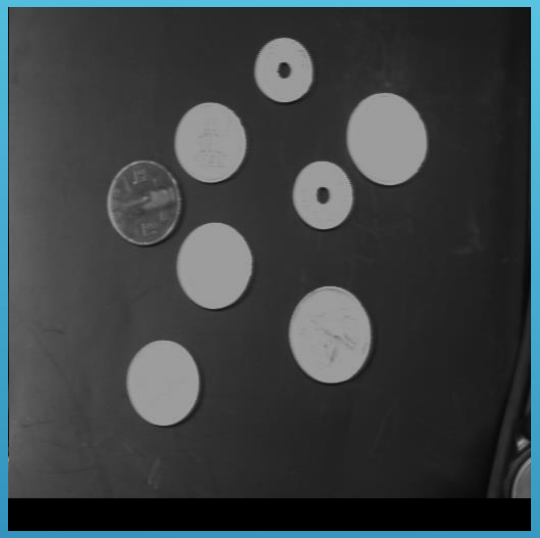
* Otzu Method의 고찰

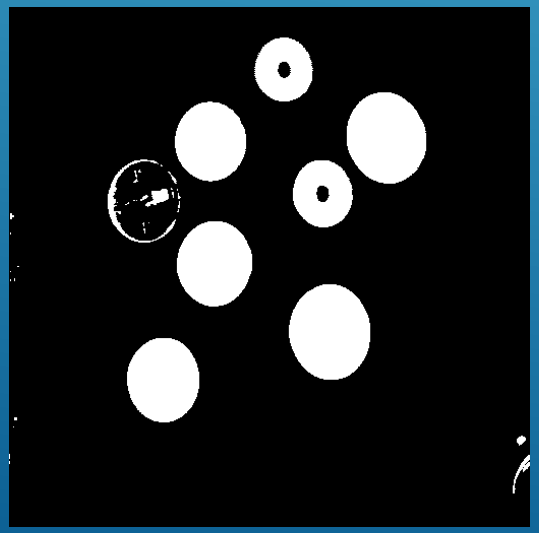
. t는 배경과 물체의 두 그룹을 나누는 역할을 함

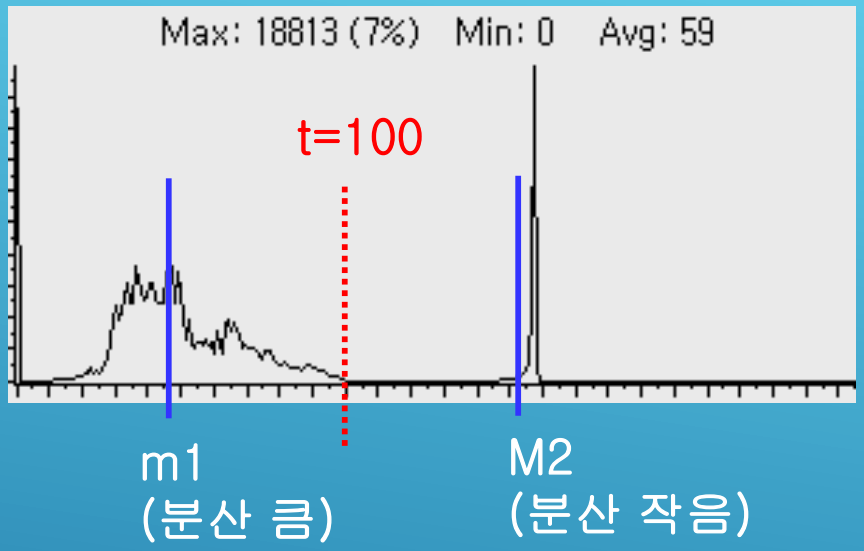
. t가 적당하게 선택되지 않아, 배경그룹에 물체피크가 포함되면 배경을 근사해 주는 정규분포의 분산값이 커지게 됨

. 따라서 t를 배경과 물체 피크가 잘 분리되게 선택해 주어야 각각을 분산값이 작아지는 정규분포로 모델링이 가능해 짐

자동 이진화







실험결과

- Optimal threshold: 100
- Group 1 Mean: 46.575
  Group 2 Mean: 152.622
- Group 1 Variance: 448.173
  Group 2 Variance: 133.493