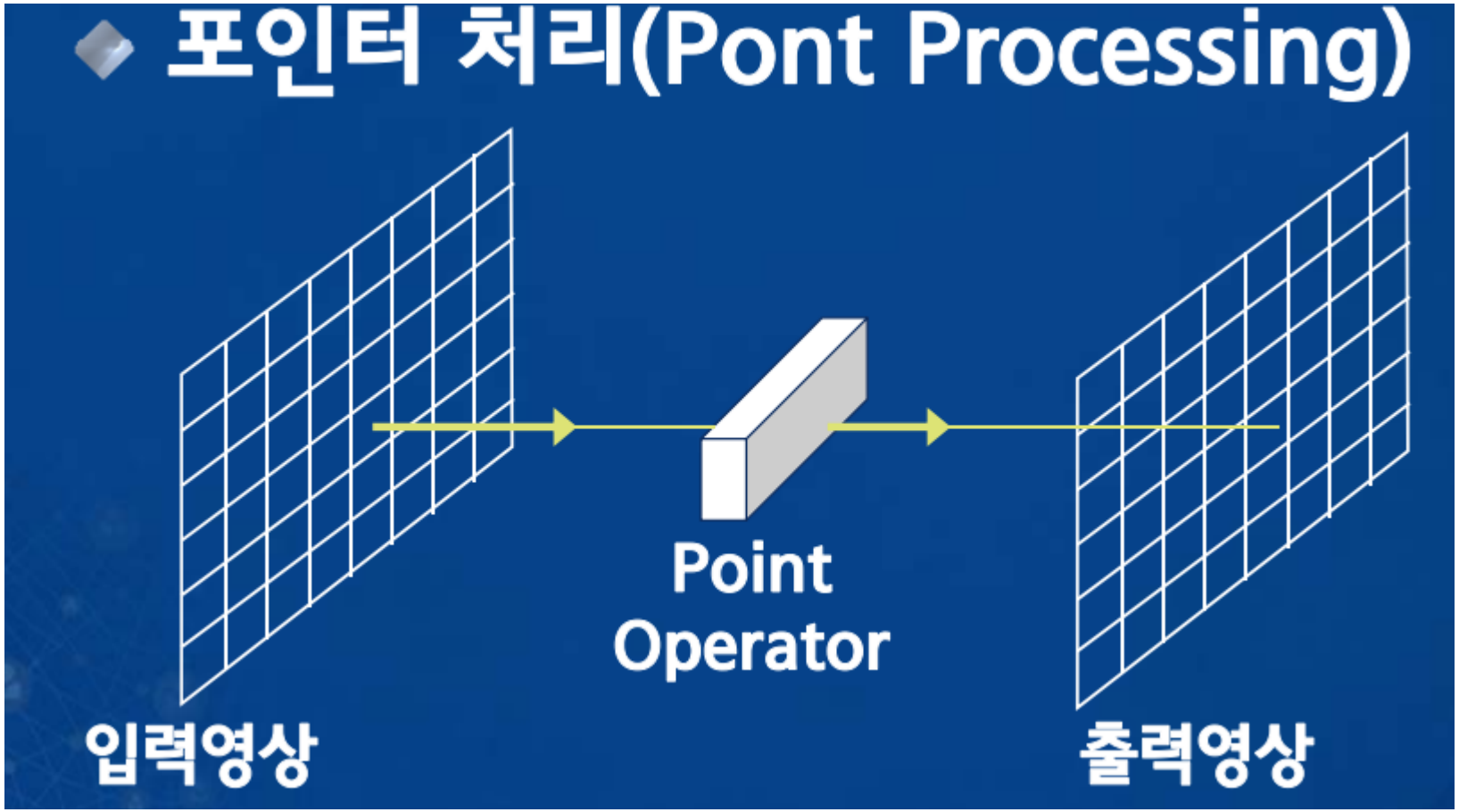
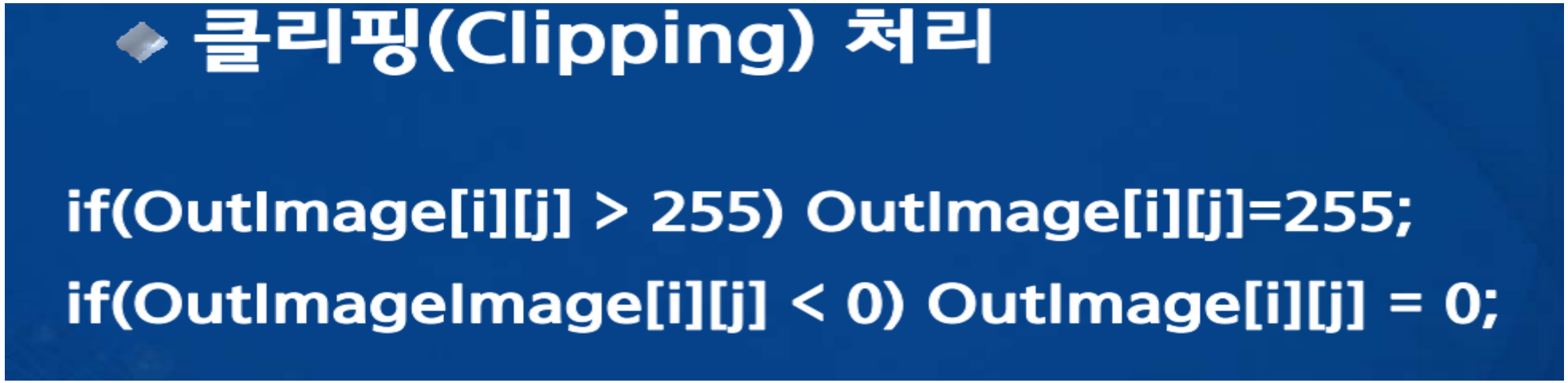
3-1. 픽셀 단위 영상처리

픽셀(포인트) 처리



어떤 위치에 있는 픽셀 값에 어떤 연산을 가해서 그 연산된 결과도 동일 위치에 저장
기본 연산 : +,-,*,/
+ : 화소값 간의 차이 유지 (밝기 유지)
* : 화소값 간의 차이 변화 (contrast(뚜렷함/모호함) 조절)

클리핑(Clipping) 처리



- Gray scale : 0~255 범위 벗어남을 방지

밝기 변화(+,-) 클리핑 처리

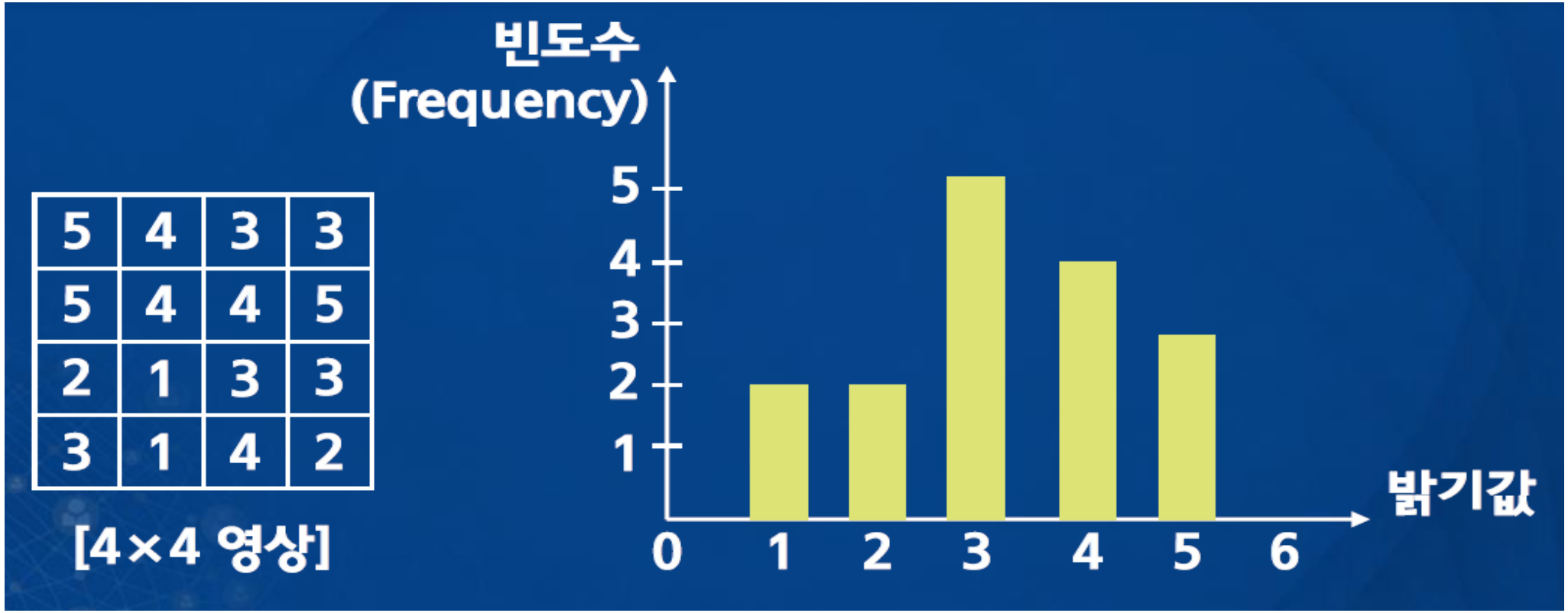
```
//밝기변화
int val;
printf("값을 입력: ");
scanf("%d", &val);
for(int i=0; i<ImgSize; i++)
    if(Image[i] + val > 255) Output[i] = 255;
    else if(Image[i] + val < 0) Output[i] = 0;
    else Output[i] = Image[i] + val;
```

대비 변화(*,/) 클리핑 처리

```
//대비변화
double val;
printf("실수 값 입력: ");
scanf("%lf", &val);
for(int i=0; i<ImgSize; i++)
    if(Image[i] * val > 255.0) Output[i] = 255;
    else Output[i] = (BYTE)(Image[i] * val);
```

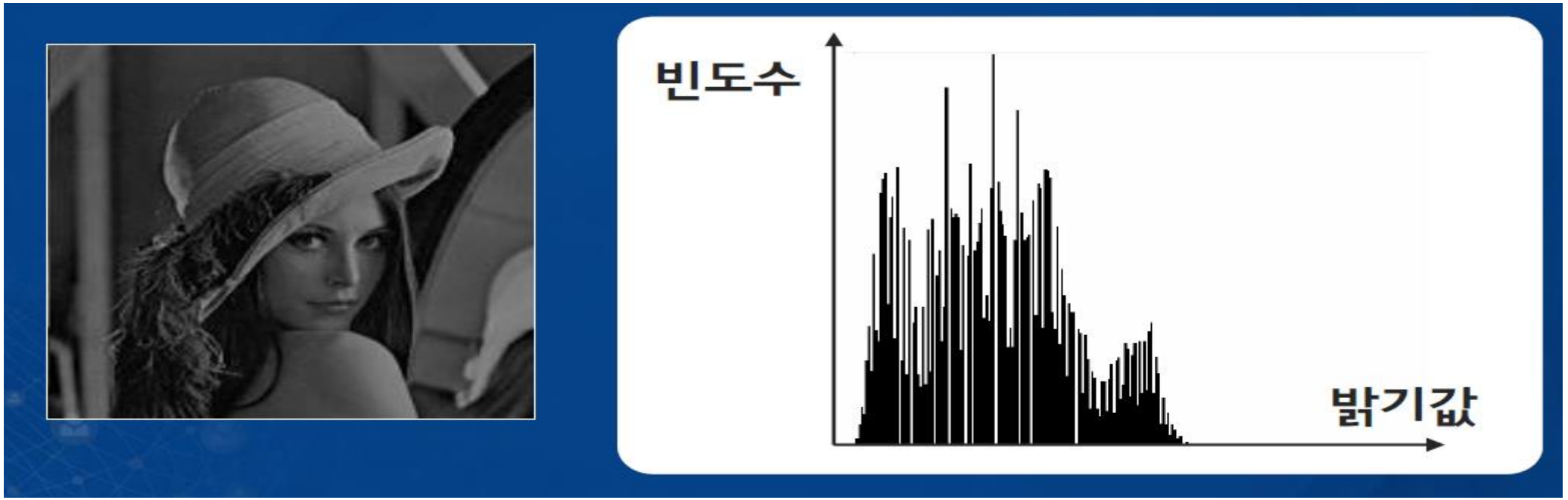
3-2. 히스토그램 기반 처리

히스토그램?



화소 밝기값에 해당하는 픽셀의 개수
영상의 상태 분석 가능

히스토그램 변화

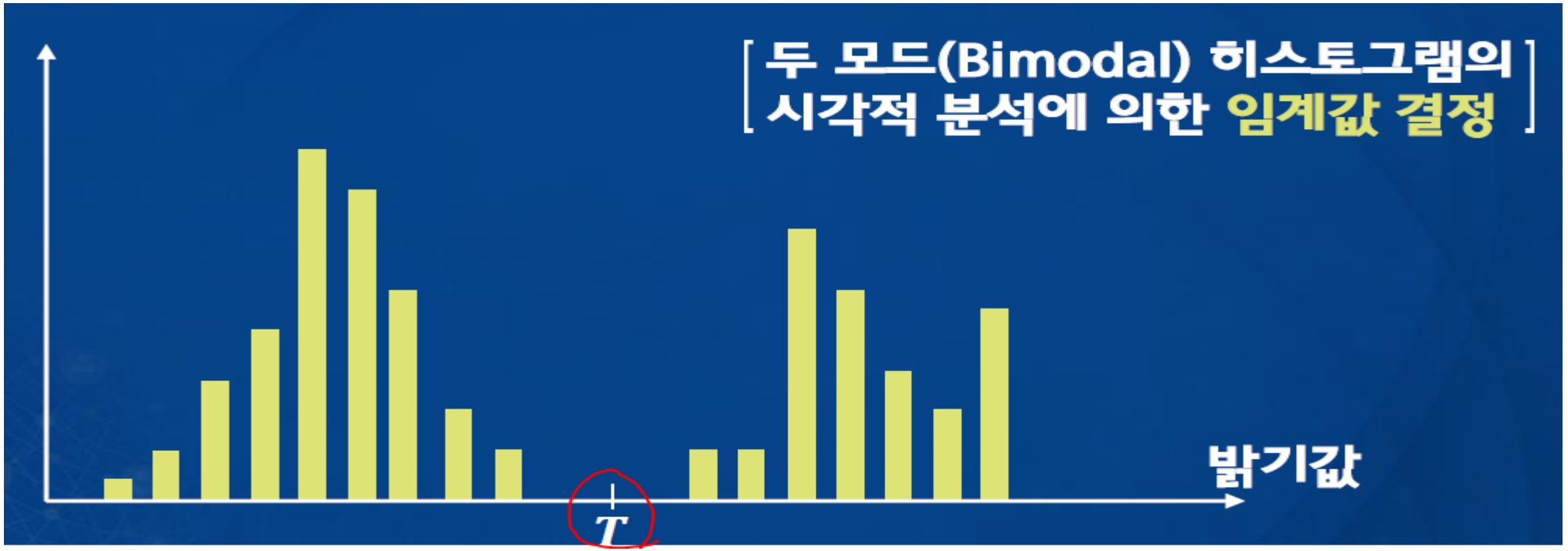


밝기 변화 : 히스토그램 왼쪽, 오른쪽 shift
Contrast 변화 : 히스토그램 scaling 변화

영상 이진화

- 영상의 화소값을 0 or 255로 표현하는 것
- “임계값”을 기준으로 임계값보다 크면 255, 작으면 0으로 표현
- 전경과 배경 분리 등에 사용

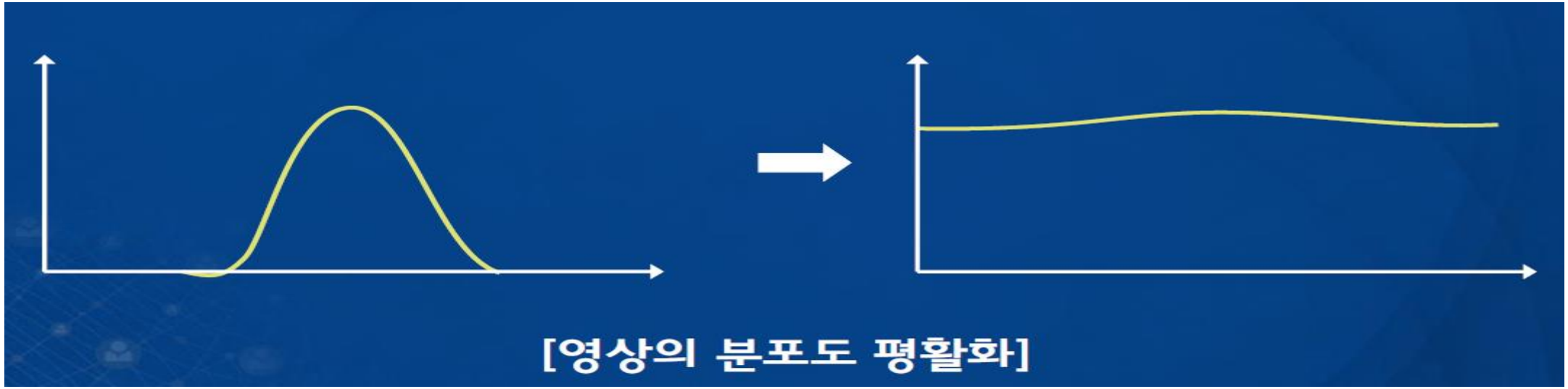
임계값 결정(bimodal)



임계값을 자동으로 설정할 수 있는 방법이 필요함
Gonzalez 방법, Otsu 방법 등 사용

히스토그램 평활화

- 영상을 개선하는 방법(Equalization)
- 밝기 분포가 치우친 영역이 없도록 히스토그램을 펼침

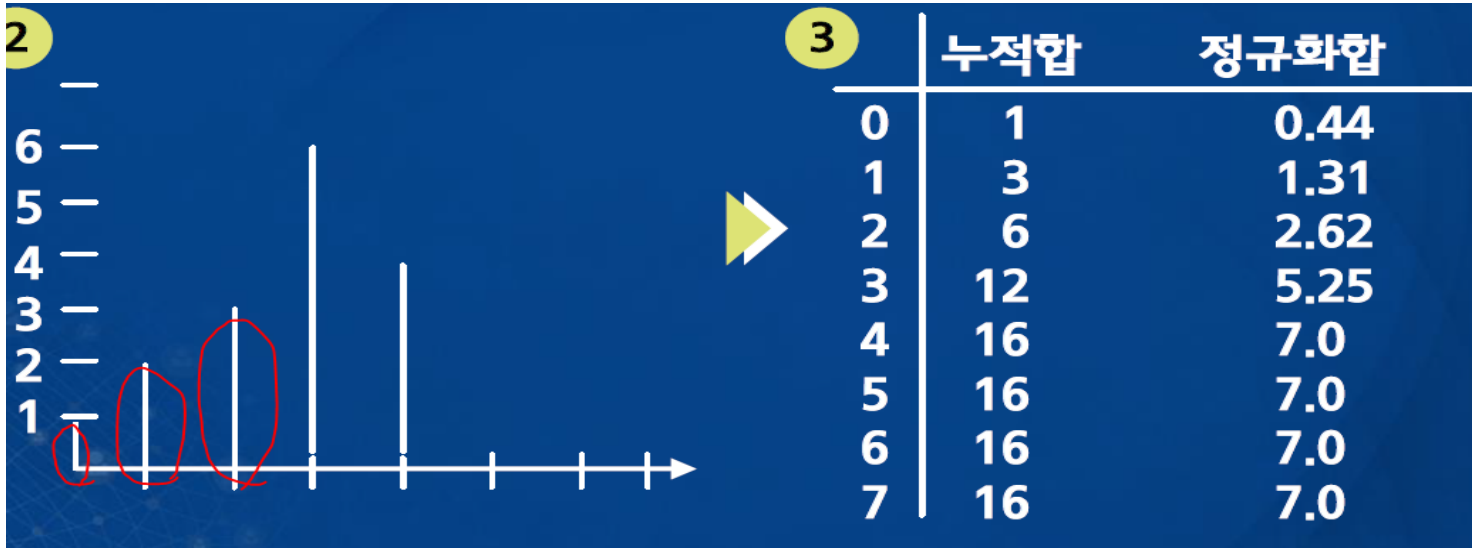


평활화 절차

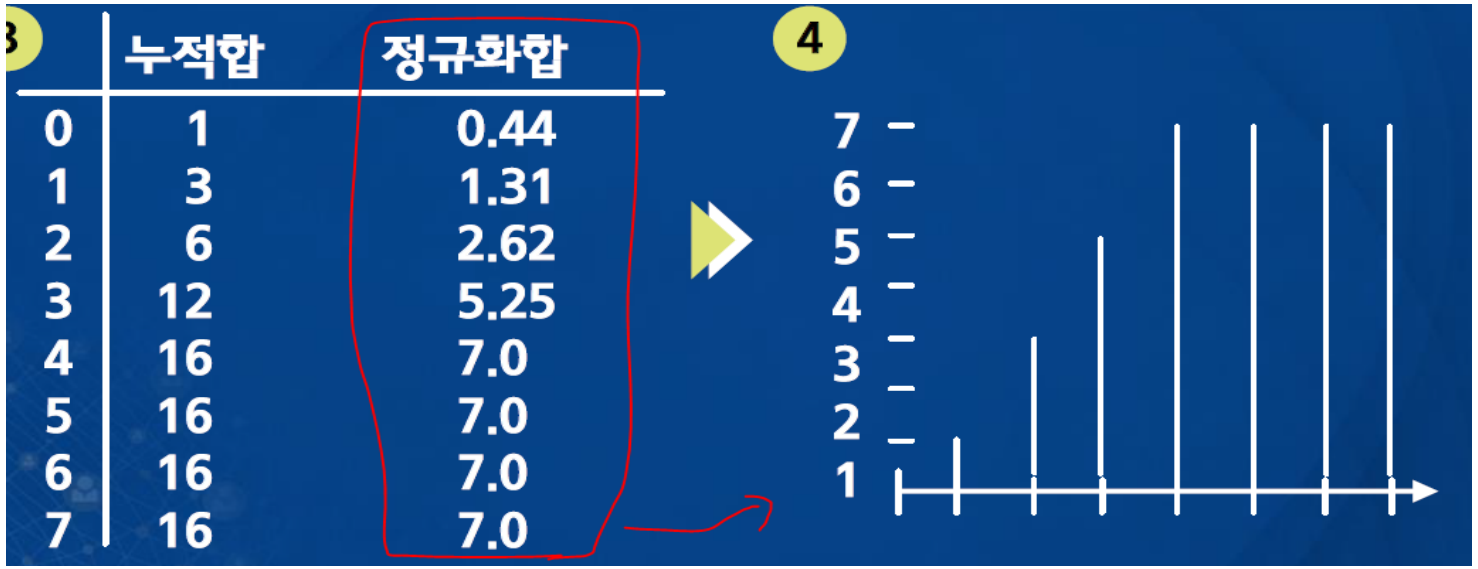
- 처음 입력영상의 밝기값에 대한 히스토그램 생성
- 생성된 히스토그램을 정규화합 히스토그램으로 변형

(unsigned character byte type, 정수만 사용)

- 정규화합 히스토그램을 이용하여 입력영상을 다시 매핑



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



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

스트레칭

- 가장 어두운 화소값과 가장 밝은 화소값을 양쪽으로 늘린 것
- 히스토그램 모양을 유지하며 linear하게 선형적으로 늘어남

스트레칭

$$OutImg[x][y] = \frac{InImg[x][y] - Low}{High - Low} \times 255$$

High : 가장 밝은 픽셀 값,
Low : 가장 어두운 픽셀 값

평활화

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

Nt : 픽셀 총 개수
H(i) : 누적 히스토그램 값
Gmax : 화소값이 가질 수 있는 가장 큰 값

개선된 스트레칭

$$OutImg[x][y] = \begin{cases} 0 & , for\ InImg[x][y] \le Low \\ \dfrac{InImg[x][y] - Low}{High - Low} \times 255 & , for\ Low \le InImg[x][y] \le High \\ 255 & , for\ InImg[x][y] > HIGH \end{cases}$$

Low보다 작으면 0

High보다 크면 255

가장 어두운, 밝은 픽셀 1개가 noise인 경우 스트레칭 실패할 수 있음.

스트레칭 코드

```
for (int i = 0; i < ImgSize; i++)
    Histo[Image[i]]++;
```

히스토그램 생성

```
int Low, High;
for (int i = 0; i < 256; i++)
{
    if (Histo[i] != 0){
        Low = i;
        break;
    }
}
```

히스토그램에서 최소값 Low에 저장

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

히스토그램에서 최대값 High에 저장

```
for (int i = 0; i < ImgSize; i++)
{
    Output[i] = (BYTE)(((Image[i] - Low) / (double)(High - Low)) * 255);
}
```

평활화 코드

```
int Histo[256] = { 0, };
int AHisto[256]; // 누적히스토그램
double NormSum[256]; // 정규화합
for (int i = 0; i < ImgSize; i++)
    Histo[Image[i]]++;
AHisto[0] = Histo[0];
for (int i = 1; i < 256; i++)
{
    AHisto[i] = AHisto[i - 1] + Histo[i];
}
```

Ahisto = 히스토그램 누적합

```
int Nt = ImgSize, Gmax = 255;
double Ratio = Gmax / (double)Nt;
for (int i = 0; i < 256; i++)
    NormSum[i] = Ratio * AHisto[i];
for (int i = 0; i < ImgSize; i++)
{
    Output[i] = (BYTE)NormSum[Image[i]];
}
```

NormSum = h(i)
소수 가능(double)

마지막에 다시 정수형으로 변환

Histogram 정보 저장

```
FILE * fp2 = fopen("Histo.txt", "wt");
for (int i = 0; i < 256; i++)
    fprintf(fp2, "%dWt%dWt%lfWn", Histo[i], AHisto[i], NormSum[i]);
fclose(fp2);
```

엑셀 등을 활용하여 그래프 통해 히스토그램 확인