디지털 영상처리 연구실 연구보고서

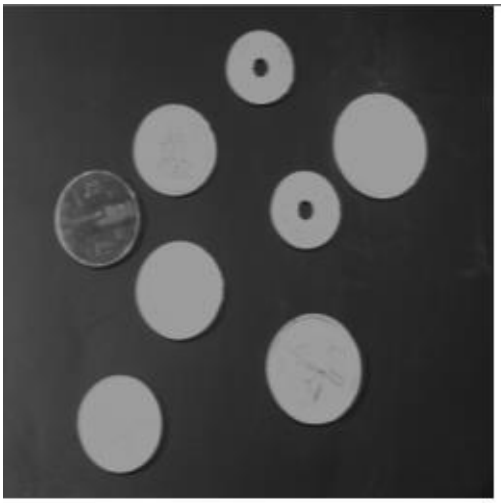
김우헌(8.11)

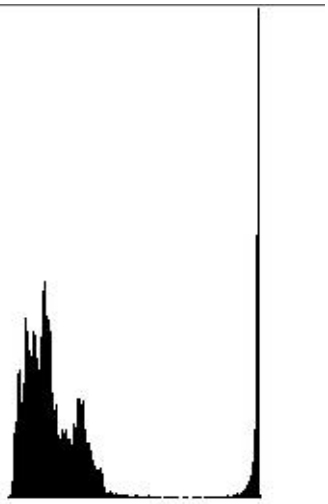
##히스토그램을 이용한 영상처리

#히스토그램 평활화

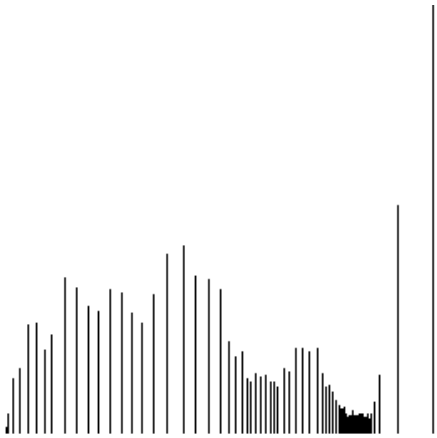
-평활화 공식(정규화합)

$$h(t) = \frac{Gmax}{Nt} H(t)$$

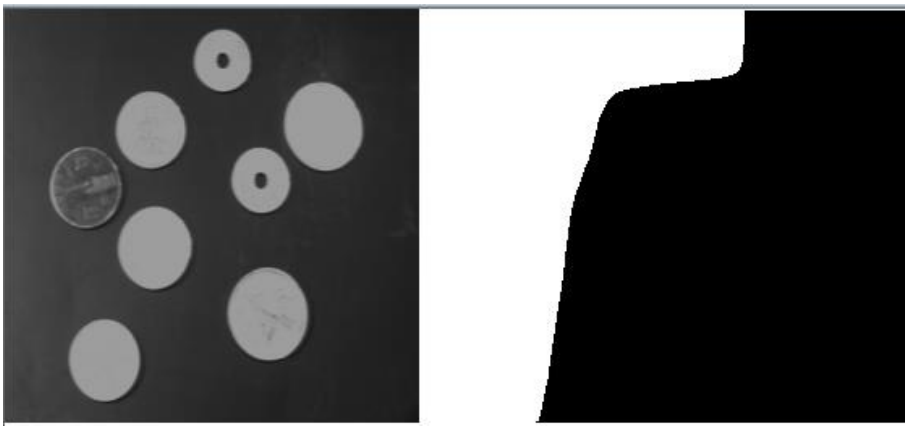




->원영상의 히스토그램



->히스토그램 평탄화 과정을 거친 히스토그램



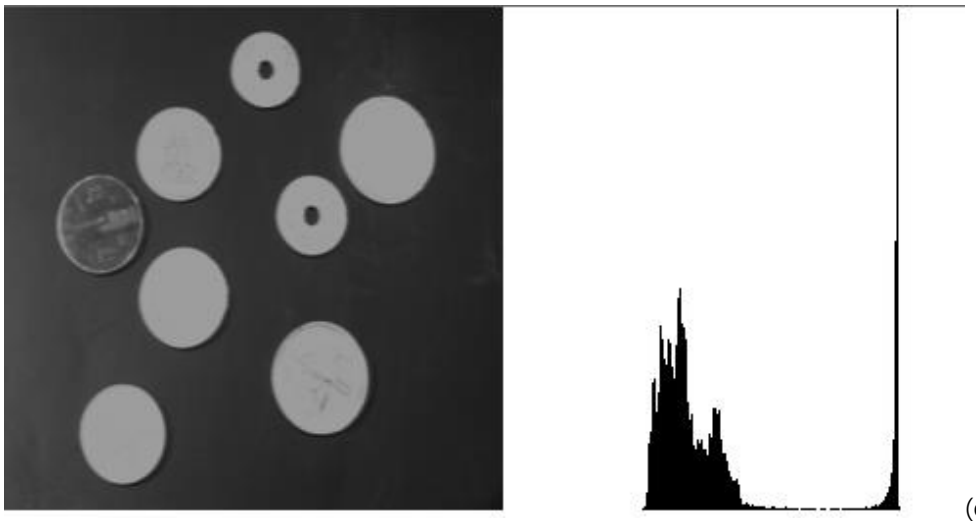
->원이미지의 누적 히스토그램

->히스토그램 평탄화 후 누적 히스토그램

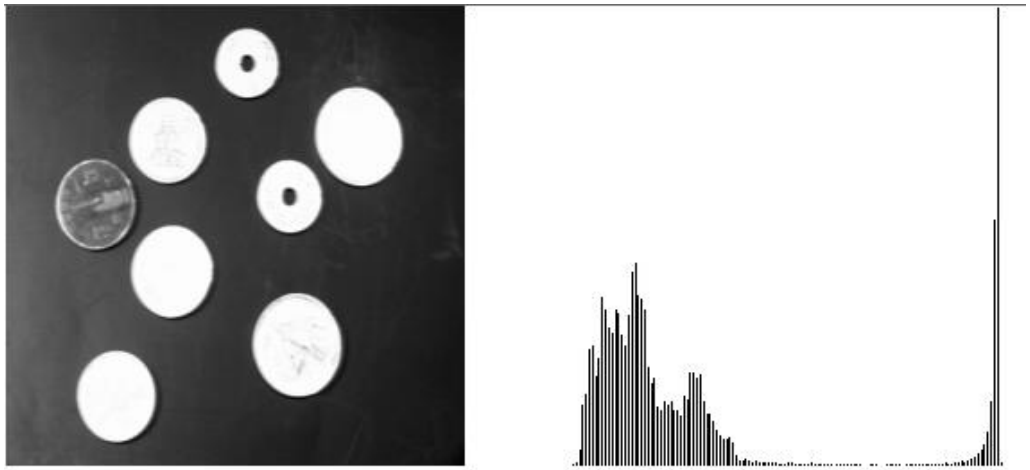
#히스토그램 스트레칭

$$new\ pixel = \frac{old\ pixel - low}{high - low} \times 255$$

-> histogram stretching 공식



(original)



(stretching)

->lenna image의 histogram

->lenna image의 히스토그램 스트레칭

#개선된 히스토그램 스트레칭

```
        for (i = 0; i < 256; i++)
          {
                runsum += histogram[i];
                if ((runsum * 100.0 / (float)(height * width)) >= lowPercent)
                {
                      lowthresh = i;
                      break;
          }
    }
```

```
for (i = 255; i >= 0; i--)
  {
        runsum += histogram[i];
        if ((runsum * 100.0 / (float)(height * width)) >= highPercent)
        {
              highthresh = i;
              break;
        }
  }
```

-> 입력받은 lowPercent, highPercent 로 lowthresh, highthresh 를 결정!

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

-> lowthresh이하의값 0으로 설정

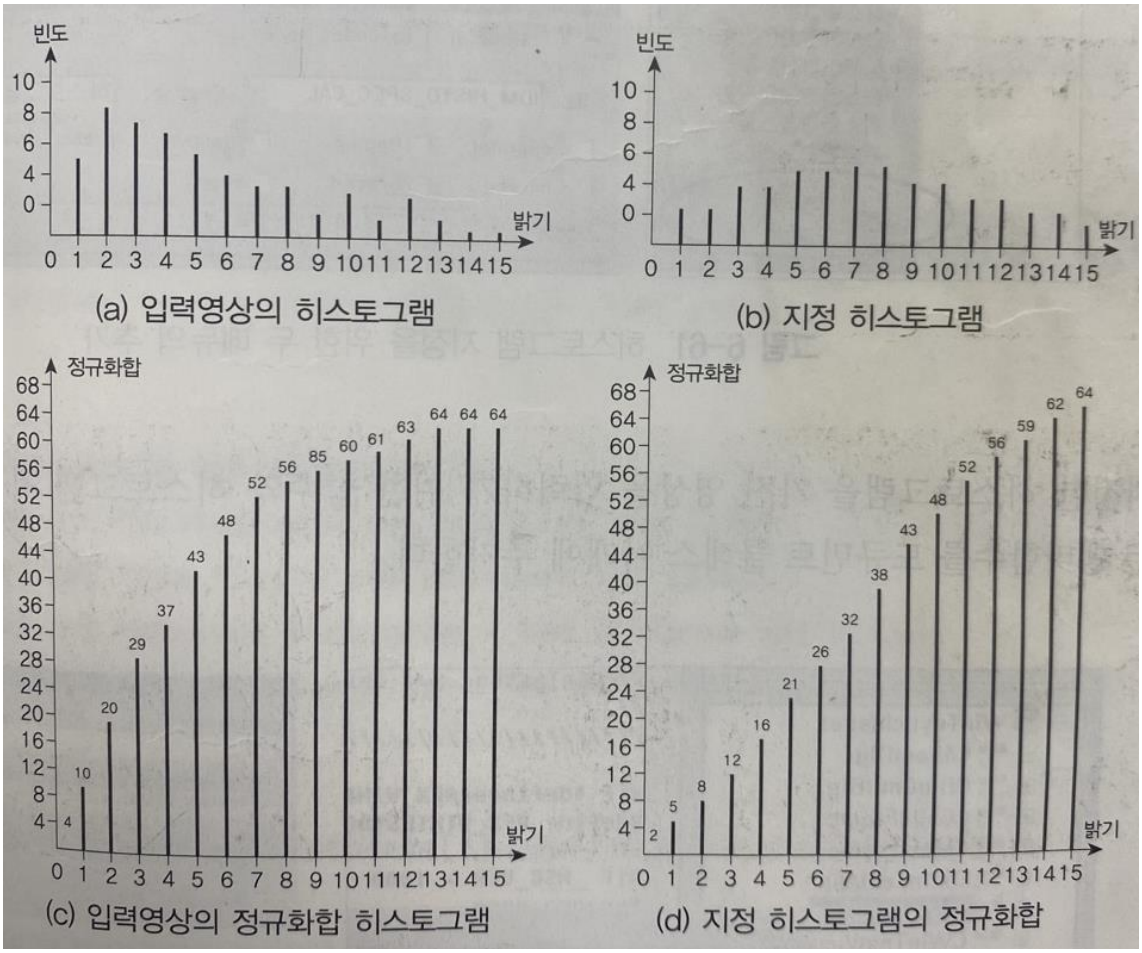
```
float scale = 255.0f / (float)(highthresh - lowthresh);
for (i = lowthresh; i <= highthresh; i++)
      LUT[i] = (unsigned char)((i - lowthresh) * scale);
```

->lowthresh와 highthresh 사이값의 스트레칭

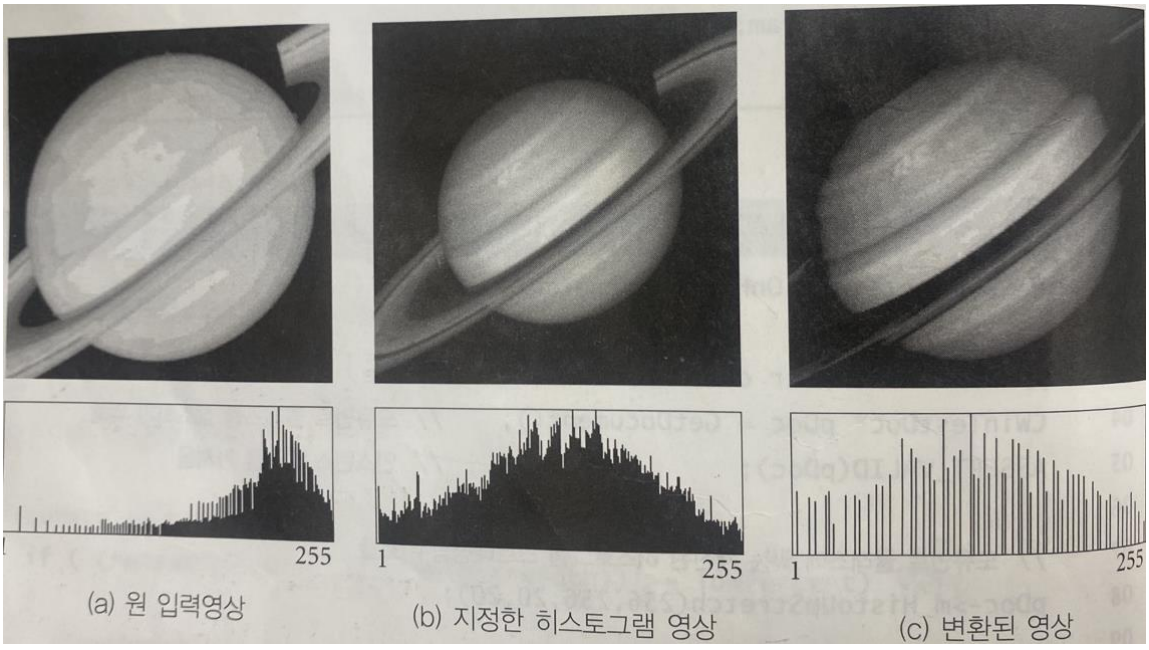
->lenna image의 개선된 히스토그램 스트레칭 (lowPercent=20%, highPercent=20%)

#히스토그램 지정

-히스토그램 지정원리



-히스토그램 지정 결과

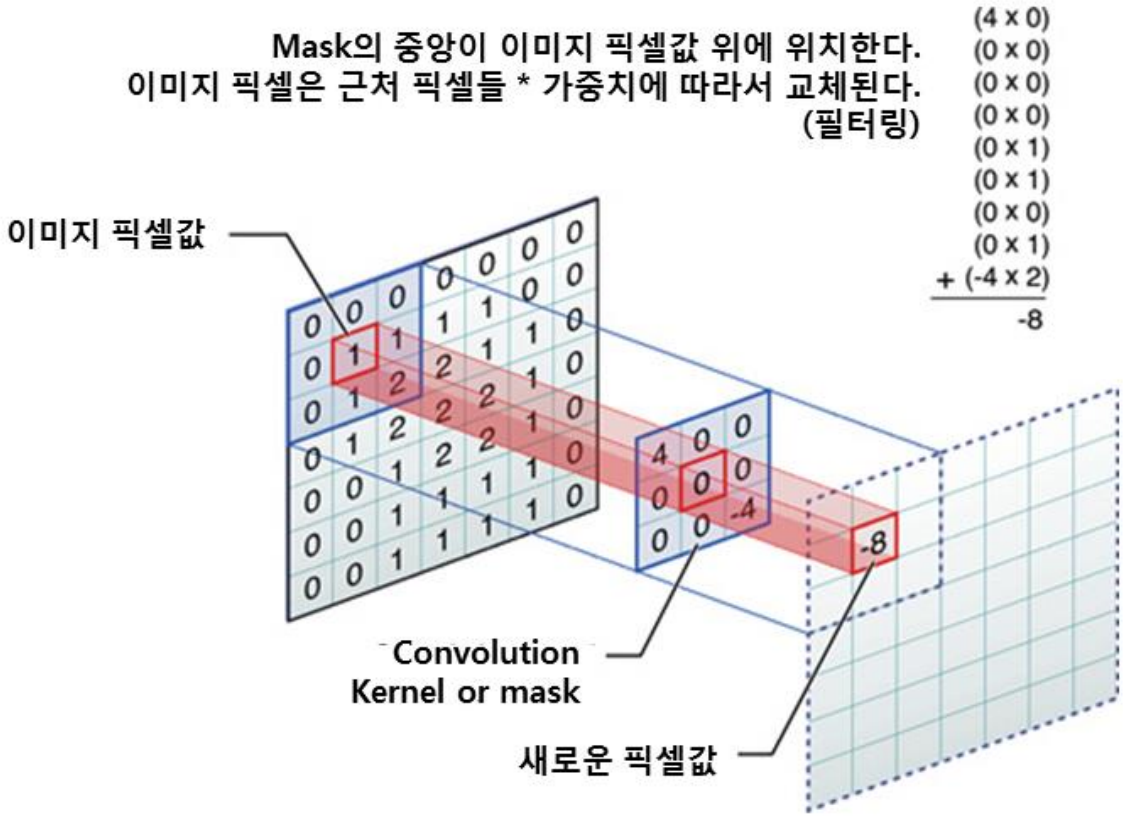


##마스크 기반 영상처리

-마스크 기반의 영상처리는 컨볼루션 연산에 의해 처리됩니다.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau$$

-디지털 영상(2차원)에서의 컨볼루션 연산



#영상 평활화

1	1	1
1	1	1
1	1	1

int MaskBox[3][3] = { {1,1,1}, {1,1,1}, {1,1,1} };

```
for (i = 1; i < heightm1; i++)
	{
		for (j = 1; j < widthm1; j++)
		{
			newValue = 0; //0으로 초기화
			for (mr = 0; mr < 3; mr++)
				for (mc = 0; mc < 3; mc++)
					newValue += (MaskBox[mr][mc] * m_inimage[i + mr - 1][j + mc - 1]);
			newValue /= 9; // 평균계산
			m_outimg[i][j] = (BYTE)newValue;//BYTE값으로 변환
		}
	}
```

1	1	1
1	1	1
1	1	1

->lenna image 의 평활화 후 모습

1	3	1
3	10	3
1	3	1

#라플라시안 마스크

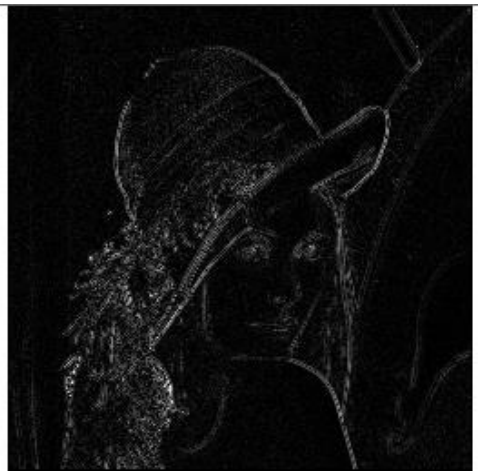
```
for (i = 1; i < heightm1; i++){
        for (j = 1; j < widthm1; j++)
            {
                    newValue = 0; //0으로 초기화
                    for (mr = 0; mr < 3; mr++)
                            for (mc = 0; mc < 3; mc++)
                                    newValue += (MaskBox[mr][mc] * m_inimage[i + mr - 1][j + mc - 1]);
                    //값을 양수로 변환
                    if (newValue < 0)
                            newValue = -newValue;
                    pTmpImg[i * width + j] = newValue;
            }}
```

-1	-1	-1
-1	8	-1
-1	-1	-1

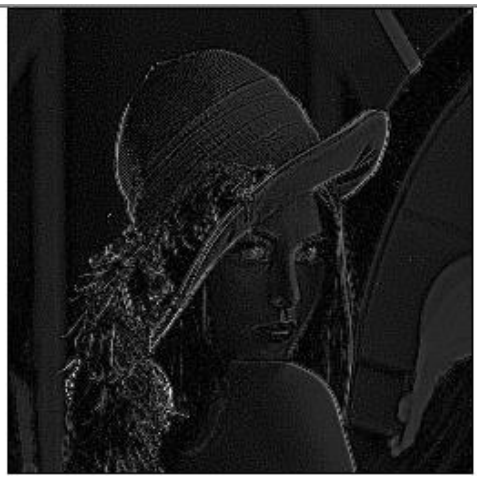
-> newValue값 (컨볼루션) 취득

```
constVal1 = (float)(255.0 / (max - min));
constVal2 = (float)(-255.0 * min / (max - min));
for (i = 1; i < heightm1; i++)
{
        for (j = 1; j < widthm1; j++)
        {
                //[min,max]사이의 값을 [0,255]값으로 변환
                newValue = pTmpImg[i * width + j];
                newValue = constVal1 * newValue + constVal2;
                m_outimg[i][j] = (BYTE)newValue;
        }
}
```

-> newValue-min값의 비 이용-> 0~255사이의 newValue 설정



-1	-1	-1
-1	8	-1
-1	-1	-1



-1	-1	-1
-1	9	-1
-1	-1	-1

#prewitt 마스크

-1	0	1
-1	0	1
-1	0	1

-> x방향prewitt mask

-1	-1	-1
0	0	0
1	1	1

-> y방향prewitt mask

```
for(i=1;i<heightm1;i++)
        for(j=1;j<widthm1;j++){
                where=i*width+j;
                constVal1=pImgPrewittX[where];
                constVal2=pImgPrewittY[where];
                if(constVal1<0)     constVal1=-constVal1;
                if(constVal2<0)      constVal2=-constVal2;
                pImgPrewittX[where]=constVal1+constVal2;
        }
```

-1	0	1
-1	0	1
-1	0	1

+

-1	-1	-1
0	0	0
1	1	1

->lenna image 의 prewitt mask후 모습

-5	0	5
-20	0	20
-5	0	5

+

-5	-20	-5
0	0	0
5	20	5

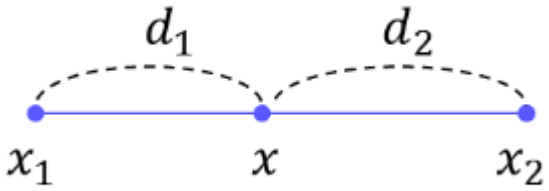
->lenna image 의 sobel mask후 모습

##영상의 기하학적 변환

-이중선형보간?

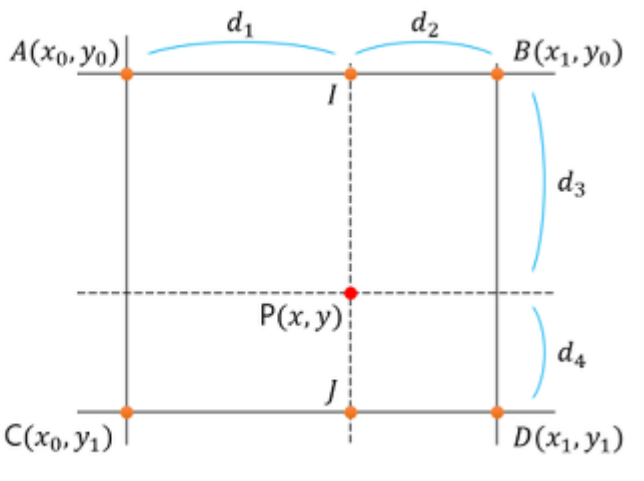
->1차원

$$f(x)=\frac{d_2}{d_1+d_2}f(x_1)+\frac{d_1}{d_1+d_2}f(x_2) \quad \text{--- (1)}$$



<그림 2>

->2차원



$$f(I) = \frac{d_1}{d_1+d_2}f(B)+\frac{d_2}{d_1+d_2}f(A)$$

$$f(J) = \frac{d_1}{d_1+d_2}f(D)+\frac{d_2}{d_1+d_2}f(C)$$

$$f(P) = \frac{d_3}{d_3+d_4}f(J)+\frac{d_4}{d_3+d_4}f(I)$$

#영상의 축소와 확대

```
r_orgr = r / zoomoutfactor;

r_orgc = c / zoomoutfactor;

i_orgr = floor(r_orgr);//예: floor(2.8)=2.0

i_orgc = floor(r_orgc);

sr = r_orgr - i_orgr;

sc = r_orgc - i_orgc;
```

➔ zoomoutfactor 설정(ex:0.7)을 통하여 r_orgr, r_orgc를 계산하고 정수값 i_orgr, i_orgc을 구합니다.

```
I1 = (float)m_inimage[i_orgr][i_orgc];

I2 = (float)m_inimage[i_orgr][i_orgc + 1];

I3 = (float)m_inimage[i_orgr + 1][i_orgc + 1];

I4 = (float)m_inimage[i_orgr + 1][i_orgc];
```

➔ 2차원 선형보간을 위한 공간을 생성합니다.

```
newValue = (BYTE)(I1 * (1 - sc) * (1 - sr) + I2 * sc * (1 - sr) + I3 * sc * sr + I4 * (1 - sc) * sr);
```

➔ 2차원 선형보간을 진행하여 newValue을 얻습니다

➔ 축소된 lenna image(zoomoutfactor =0.7)


➔ 확대된 lenna image(zoomoutfactor =1.3)

#영상의 회전

-역방향 매핑 이용

$$\begin{pmatrix} x_{org} \\ y_{org} \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{new} - c_x \\ y_{new} - c_y \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix}$$

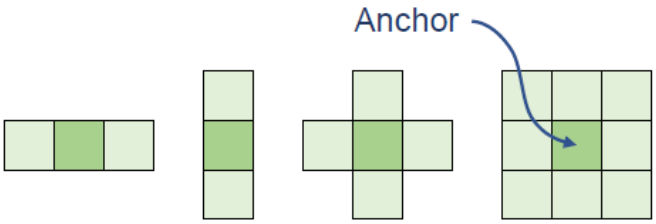
```
for (r = 0; r < height; r++)
        for (c = 0; c < width; c++)
        {
                cosAngle = (float)cos(rotationAngleRad);
                sinAngle = (float)sin(rotationAngleRad);
                //회전전의 원 이미지상의 좌표 구함
                r_orgc = cosAngle * (c - center_c)   + sinAngle * (r - center_r)   + center_c;//가로
                r_orgr = - sinAngle * (c - center_c) + cosAngle * (r - center_r)   + center_r;//세로

                i_orgr = floor(r_orgr);//예: floor(2.8)=2.0
                i_orgc = floor(r_orgc);
                sr = r_orgr - i_orgr;
                sc = r_orgc - i_orgc;
```

➔ Lenna image를 기준점(256/2,256/2)을 기준으로 35도 회전

➔ Lenna image를 기준점(0,0)을 기준으로 45도 회전

##모폴로지

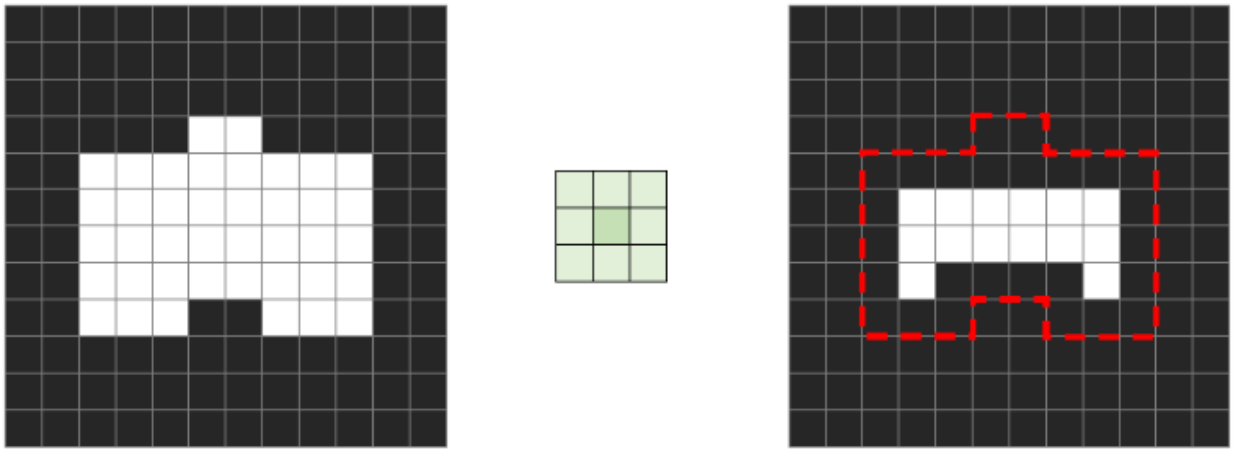
- 다양한 구조요소 사용



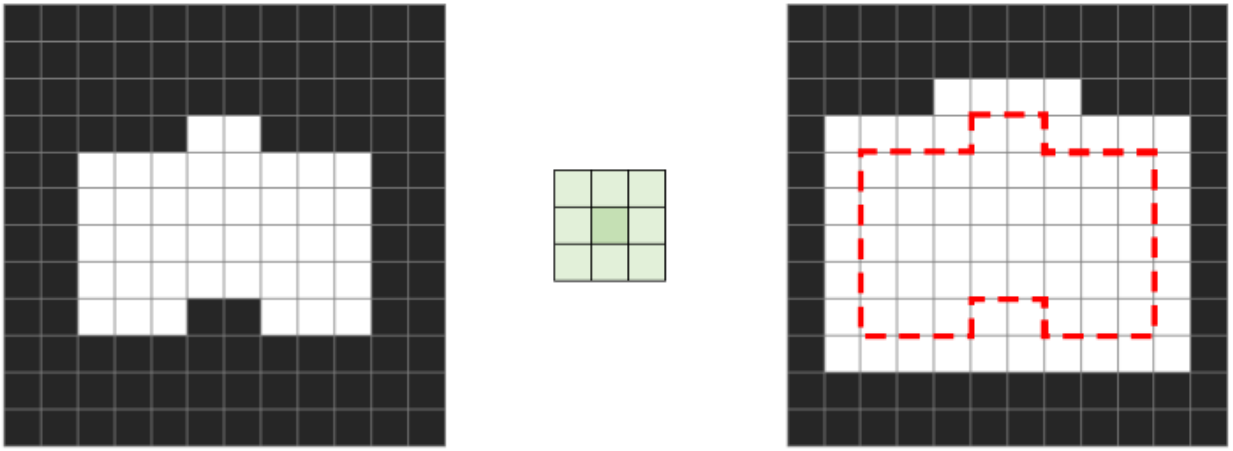
#이진 모폴로지 연산

★이진 모폴로지는 이치화된 영상을 사용하거나 이치화한 후 모폴로지 연산하는 것!

- 구조요소의 모든 요소가 영역내에 존재 -> 1 ->침식(erosion)



- 구조요소의 요소의 값 중 하나라도 존재 시 -> 1 ->팽창(dilation)

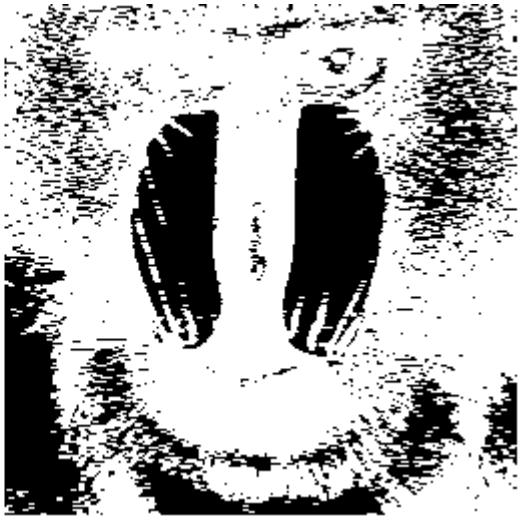


#모폴로지 침식

```
pDoc->m_AllocStructureElementBinary(4);
pDoc->m_SetStructureElementBinary(0, 4, 0);
pDoc->m_SetStructureElementBinary(1, 0, -1);
pDoc->m_SetStructureElementBinary(2, 0, 0);
pDoc->m_SetStructureElementBinary(3, 0, 1);
```

➔ 구조요소 지정(크기4) = (0,-1)(0,0)(0,1)

```
for (r = 0; r < height; r++)
        for (c = 0; c < width; c++)
        {
                flagPassed = 1;
                for (i = 1; i < m_pSEBinary[0].row; i++)
                {
                        mx = c + m_pSEBinary[i].col;
                        my = r + m_pSEBinary[i].row;//범위 검사
                        if (mx >= 0 && mx < width && my >= 0 && my < height)
                         if (m_inimage[my][mx] == BACKGROUND)//
                    //하나라도 BACKGROUND=255값을 포함하면 제일 안쪽 for loop를 빠져나감.
                        {
                                flagPassed = 0;
                                break;
                                        }
                 }
                        if (flagPassed)
                        m_outimg[r][c] = FOREGROUND; //매핑
        }
```



➔ 이진화된 원숭이image를 침식 연산한 모습

#모폴로지 팽창

```
for (r = 0; r < height; r++)
        for (c = 0; c < width; c++)
          {
                flagPassed = 0;
                for (i = 1; i < m_pSEBinary[0].row; i++)
            {
                    mx = c + m_pSEBinary[i].col;
                    my = r + m_pSEBinary[i].row;
                    if (mx >= 0 && mx < width && my >= 0 && my < height)
                    if (m_inimage[my][mx] == FOREGROUND)
                  //하나라도 FOREGROUND=0값을 포함하면 제일 안쪽 for loop를 빠져나감.
                    {   flagPassed = 1;
                        break;
                    }
              }
                    if (flagPassed)
                       m_outimg[r][c] = FOREGROUND;
              }
```

➔ 이진화된 원숭이image를 팽창 연산한 모습

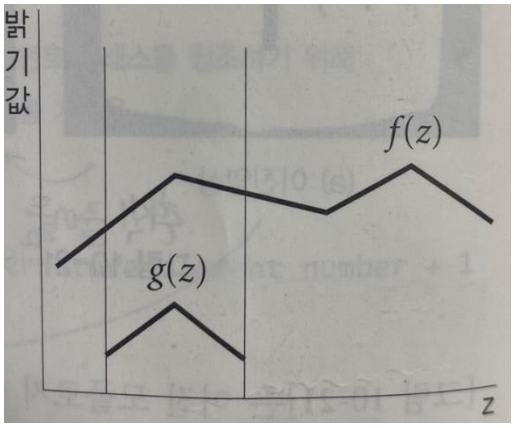
#제거,채움연산

➔ 팽창후 침식=제거연산

-> 침식후 팽창=채움연산

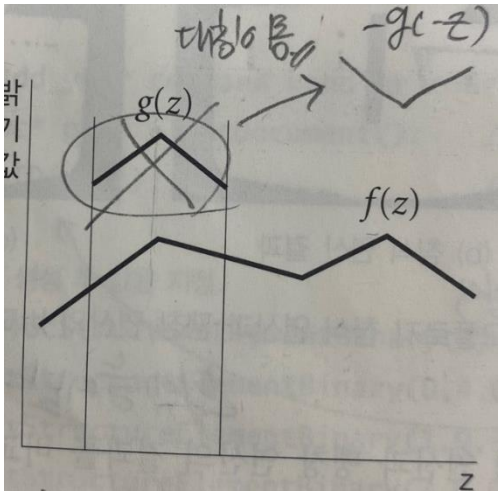
#그레이 영상의 모폴로지

-그레이 영상의 침식



G(x)<<F(x)의 최대값

-그레이 영상의 팽창



-G(-x)>>F(x)의 최소값

-구조이미지설정

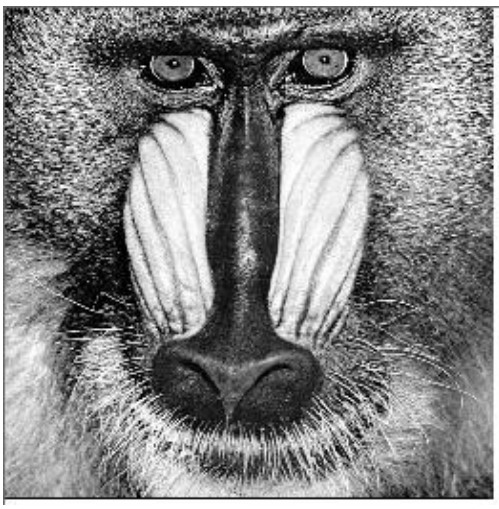
```
pDoc->m_AllocStructureElementGray(6);
pDoc->m_SetStructureElementGray(0, 6, 0, 0);
pDoc->m_SetStructureElementGray(1, -1, 0, 1);
pDoc->m_SetStructureElementGray(2, 0, -1, 1);
pDoc->m_SetStructureElementGray(3, 0, 0, 2);
pDoc->m_SetStructureElementGray(4, 0, 1, 1);
pDoc->m_SetStructureElementGray(5, 1, 0, 1);
```

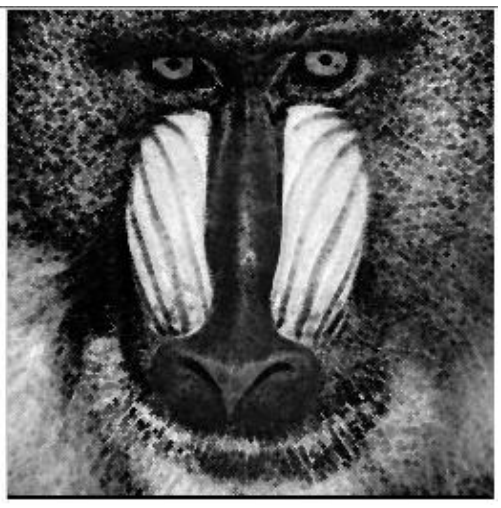
	(-1,0,1)	
(0,-1,1)	(0,0,2)	(0,1,1)
	(1,0,1)	

-그레이 영상의 침식연산 과정 G(x)<<F(x)의 최대값

```
for (r = 0; r < height; r++)
        for (c = 0; c < width; c++)
        {
                min = m_inimage[r][c];
                for (i = 1; i < m_pSEGray[0].row; i++)
                {
                        mx = c + m_pSEGray[i].col;
                        my = r + m_pSEGray[i].row;
                        //범위 검사
                        if (mx >= 0 && mx < width && my >= 0 && my < height)
                        {
                                diff = m_inimage[my][mx] - m_pSEGray[i].grayval;
                                if (diff < min)
                                        min = diff;
                        }
                }
                pTmpImg[r * width + c] = min;
        }
```

->min으로 침식연산후 나머지과정은 이진영상의 디스플레이 과정과 동일

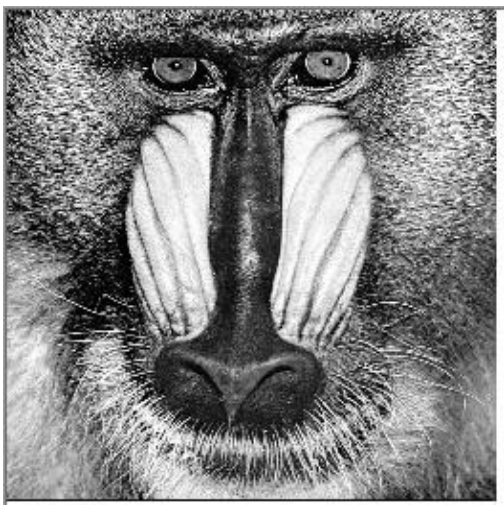


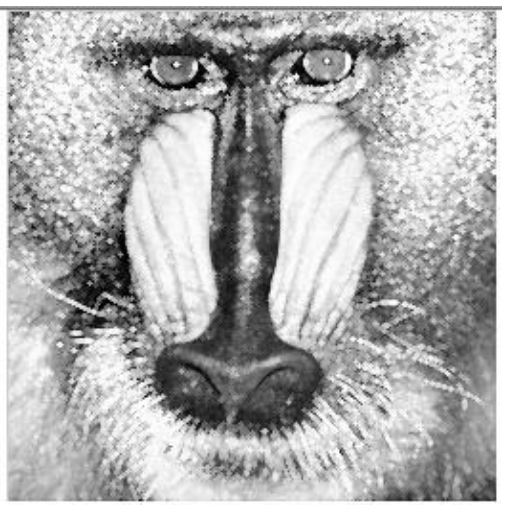


->밝은 영역의 침식

-그레이 영상의 팽창 -G(-x)>>F(x)의 최소값

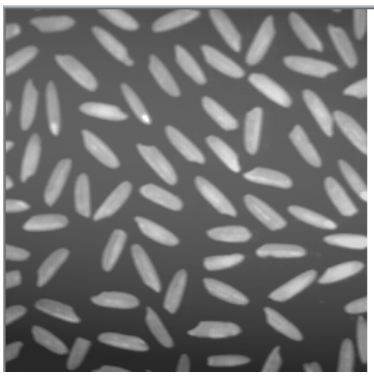
```
for (r = 0; r < height; r++)
        for (c = 0; c < width; c++)
        {
                max = m_inimage[r][c];
                for (i = 1; i < m_pSEGray[0].row; i++)
                {
                        mx = c - m_pSEGray[i].col;
                        my = r - m_pSEGray[i].row;
                        //범위 검사
                        if (mx >= 0 && mx < width && my >= 0 && my < height)
                        {
                                sum = m_inimage[my][mx] + m_pSEGray[i].grayval;
                                if (sum > max)
                                        max = sum;
                        }
                }
                pTmpImg[r * width + c] = max;
        }
```

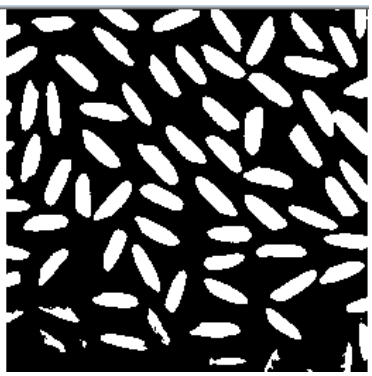




->밝은 영역의 팽창

#top hat 연산





- Otzu 이치화 =배경이 불균일 하여 이치화 하기 어려움

➔ 흑백영상에서 열림,닫힘 연산을 실행하고 원영상과의 차를 이용해 top hat수행

-밝은 물체 추출시 TOPHAT(A,B)=A-MAX(MIN(A))

-어두운 물체 추출시 TOPHAT(A,B)=A-MIN(MAX(A))

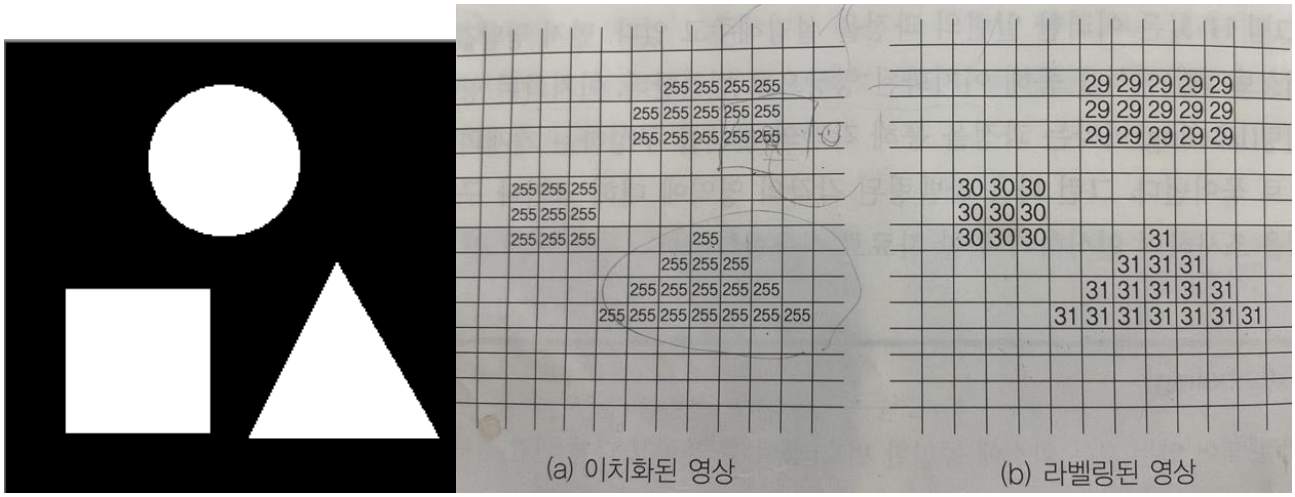
-MIN(A),MAX(A)

```
for(i=0; i<height; i++)
{               int index2 = i*width;
                for(j=0; j<width; j++)
                {           int minVal = 100000;
                            for(k=-r; k<=r; k++)     //r=30으로 설정
                            {           if(i+k<0 || i+k>=height) continue;
                                        int index1 = (i+k)*width;
                                        for(l=-r; l<=r; l++)
                                        {           if(j+l<0 || j+l >=width) continue;
                                                    uchar imVal = orgImg[index1+j+l];
                            if(imVal<minVal) minVal = imVal;//              if(imVal>maxVal) maxVal = imVal;
                                        }
                            }
                            outImg[index2+j] = (uchar)minVal;//             outImg[index2+j] = (uchar)maxVal;
                }
    }
```

원본 IMAGE(A) - MAX(MIN(A)) = A- MAX(MIN(A))

##이진영상을 이용한 영상인식

#라벨링



-라벨링 단계

	255	
255	255	255

→

	31	
255	255	255

→

		31`
	255	255
255	255	255

→

		31`
	31	255
255	255	255

#GLASSFIRE 알고리즘

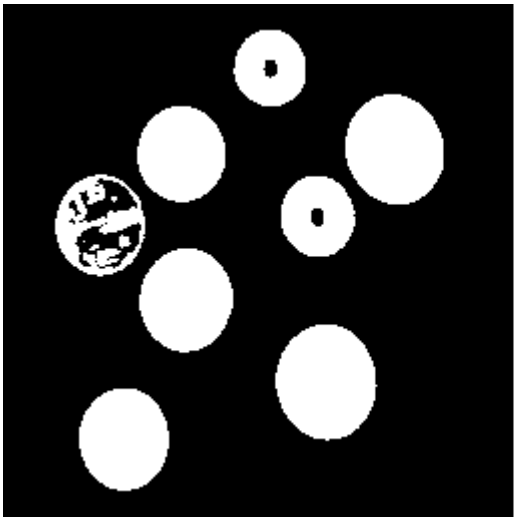
-> 자가호출 방식 사용

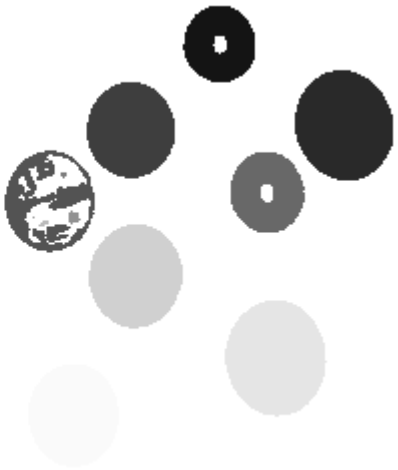
```
for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++)
        {
            if (m_outimg[i][j] == 255 && coloring[i * width + j] == 0)
            {
                curColor++;
                grass(coloring, height, width, i, j, curColor);
            }
        }
    }
```

➔ 255로 시작되는 지점과 지나간점을 표시하는 coloring INDEX가0인 지점을찾아 curColor INDEX를 카운트하고 GRASS함수 호출

-GRASS함수

```
for (k = i - 1; k <= i + 1; k++)
    {       for (l = j - 1; l <= j + 1; l++)
        {       if (k < 0 || k >= height || l < 0 || l >= width) continue;
            index = k * width + l;
            if (m_outimg[k][l] == 255 && coloring[index] == 0
            {       coloring[index] = curColor; )//방문한 지점을 마킹
                grass(coloring, height, width, k, l, curColor);
            }
        }
    }
```



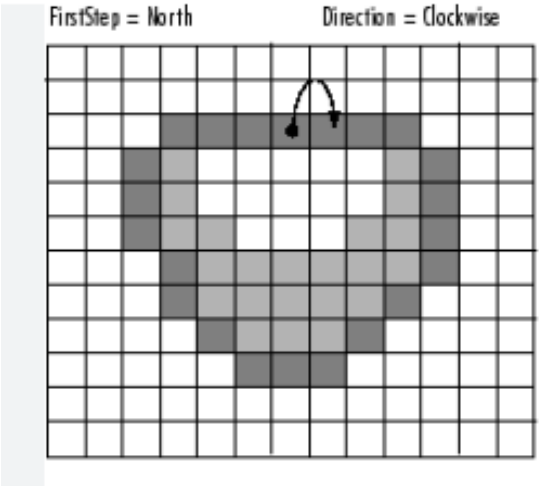


->이진화IMAGE를 라벨링한 모습

#반복문을 사용한 GLASSFIRE 알고리즘

```
while(1) {

GRASSFIRE:

        for(m=r-1; m<=r+1; m++)

         {      for(n=c-1; n<=c+1; n++)

             {      if(m<0 || m>=height || n<0 || n>=width) continue;

                    if((int)m_InImg[m][n]==255 && coloring[m*width+n]==0)

                              {   coloring[m*width+n]=curColor;   // 현재 라벨로 마크

                                  if(push(stackx,stacky,(short)m,(short)n,&top)==-1) continue;//SIZE초과시(-1)

                                        r=m; //기준변경

                                        c=n;

                                        area++;

                                        goto GRASSFIRE;

                               }

                    }

           }

           if(pop(stackx,stacky,&r,&c,&top)==-1) break;          //TOP=0    -> RETURN(-1)

        }
```

#영역 경계의 추적



-경계추적

3	4	5
2	X	6
1	0	7

->관심픽셀의 주위 픽셀번호

->처음시작은 4번부터

-> N`=(N+5)&7;

```
const POINT nei[8] =                    // clockwise neighbours
        {
                {1,0}, {1,-1}, {0,-1}, {-1,-1}, {-1,0}, {-1,1}, {0,1}, {1,1}
        };
```

->nei[] 를 통한 경계추적 박스 설정

(-1,-1)	(-1,0)	(-1,1)
(0,-1)	X	(0,1)
(1,-1)	(1,0)	(1,1)

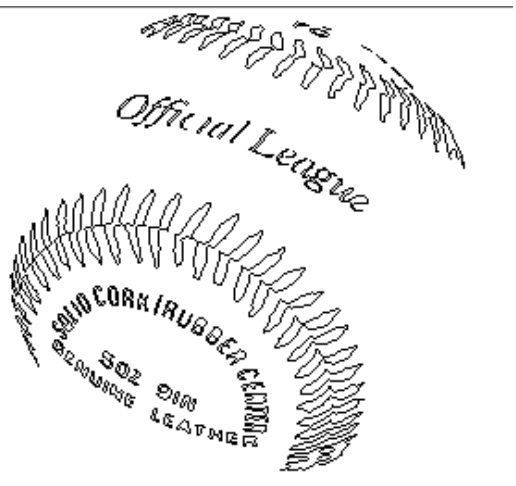
```
do
    {
            for (k = 0; k < 8; k++, n = ((n + 1) & 7))
              {
                    short   u = (short)(x + nei[n].x);
                    short   v = (short)(y + nei[n].y);
                    if (u < 0 || u >= height || v < 0 || v >= width) continue;
                    if (m_inimage[u][v] == c0) break; // 관심점의 주위를 돌다가 같은 밝기의

                }
            if (k == 8) break;      // 고립점(영역내 추적할영역의 화소가 하나)
            visited[x * width + y] = 255;   // 방문한 점으로 마크
            xchain[border_count] = x;
            ychain[border_count++] = y;   // border_count++통한 경계면의 좌표를 0부터 증가시킴
            if (border_count >= 10000) break;
            x = x + nei[n].x;
            y = y + nei[n].y;           //중심이동
            if (n % 2 == 1) diagonal_count++;
            n = (n + 5) & 7;            //n값설정
    }while (!(x == x0 && y == y0));//처음기준좌표
if (k == 8) continue;      // 고립점(영역내 추적할영역의 화소가 하나)
```

```
            stBorderInfo[numberBorder].x = new short[border_count];
            stBorderInfo[numberBorder].y = new short[border_count];
// border_count는 경계면의 개수
for (k = 0; k < border_count; k++)
                                {
                                        stBorderInfo[numberBorder].x[k] = xchain[k];
                                        stBorderInfo[numberBorder].y[k] = ychain[k];
                                }
                                stBorderInfo[numberBorder].n = border_count;
                                stBorderInfo[numberBorder++].dn = diagonal_count;
```

->구조체의[numberBorder] 즉 경계면의 개수가 변수가 됨

```
for (k = 0; k < numberBorder; k++)
  {     for (int i = 0; i < stBorderInfo[k].n; i++)
          {          x = stBorderInfo[k].x[i];
                     y = stBorderInfo[k].y[i];
                     m_outimg[x][y] = 0;
          }
  }
```

-> stBorderInfo 구조체를 이용하여 경계면 0으로 설정



->경계추출