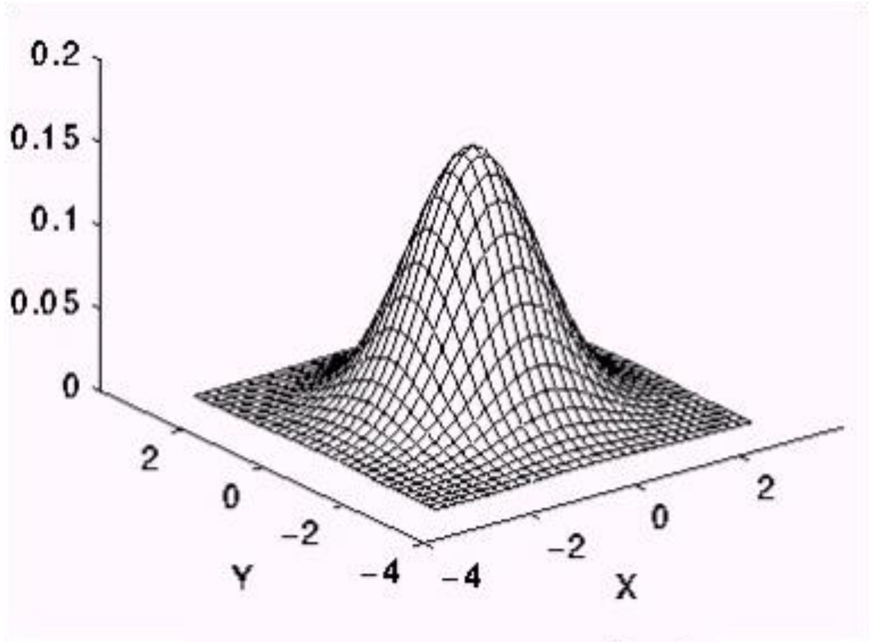
영상 직선 추출

20131356 추교준

가우시안 블러링



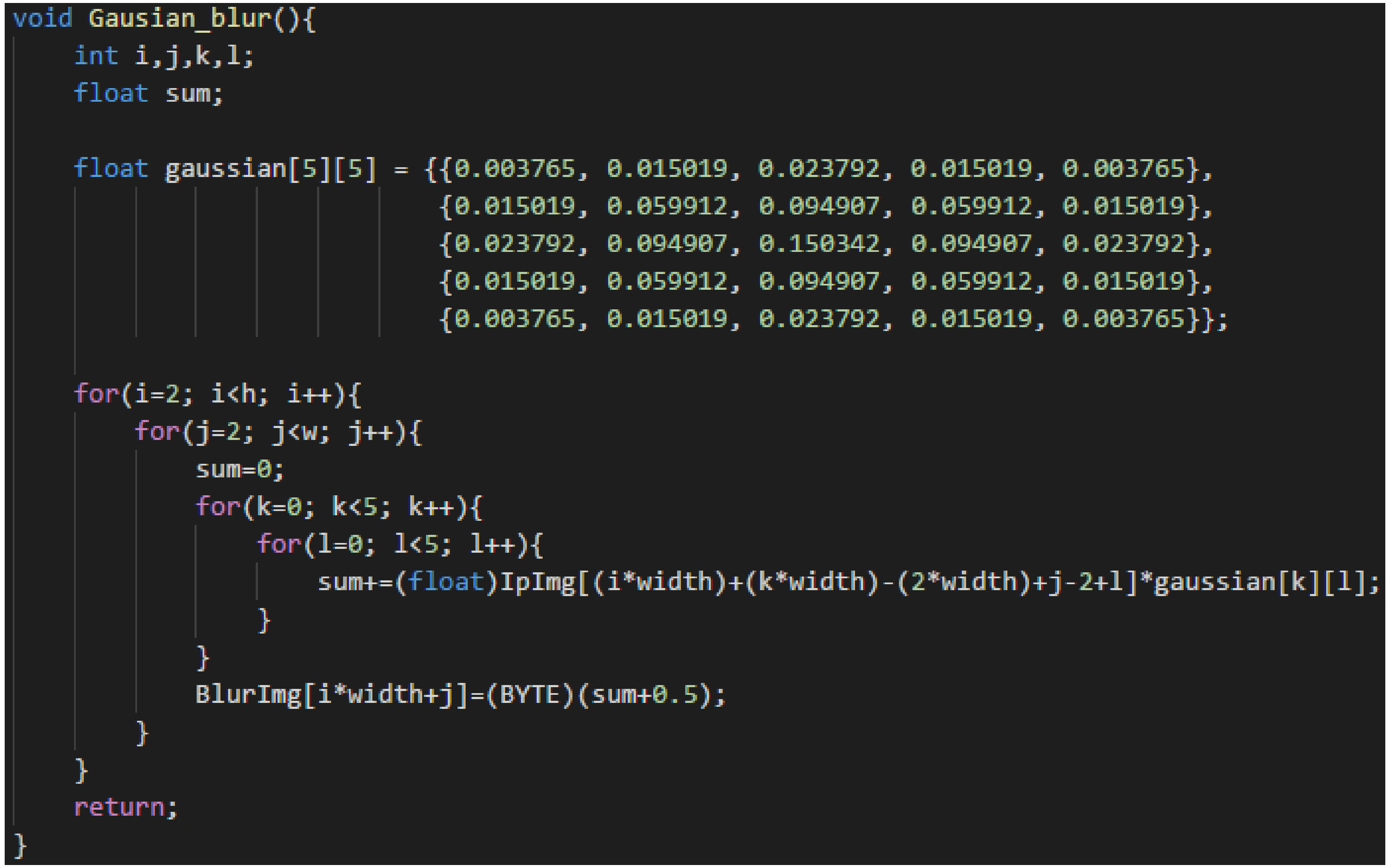
$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\frac{-(x^2+y^2)}{2\sigma^2}$$

가우시안 분포

0.003	0.015	0.238	0.015	0.003
0.015	0.060	0.094	0.060	0.015
0.023	0.094	0.15	0.094	0.023
0.015	0.060	0.094	0.060	0.015
0.003	0.015	0.238	0.015	0.003

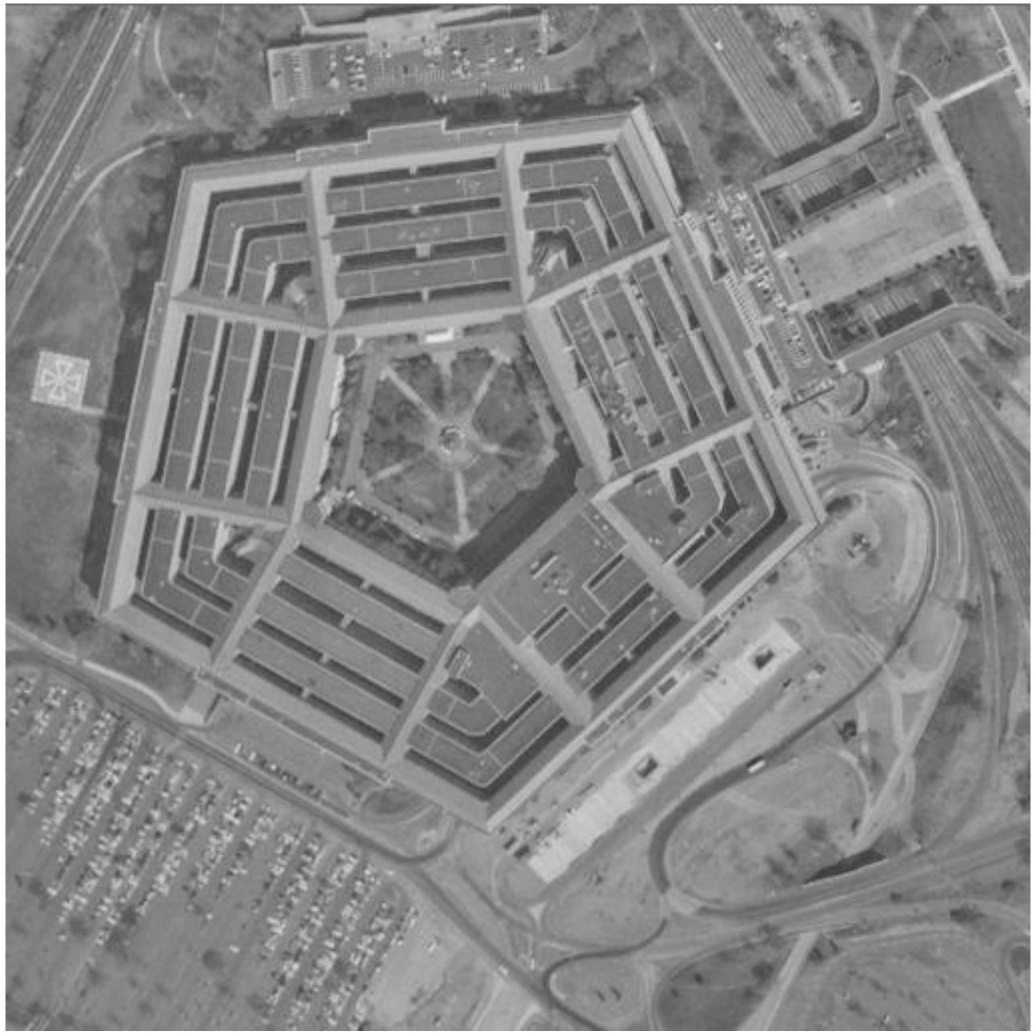
가우시안 분포 상수

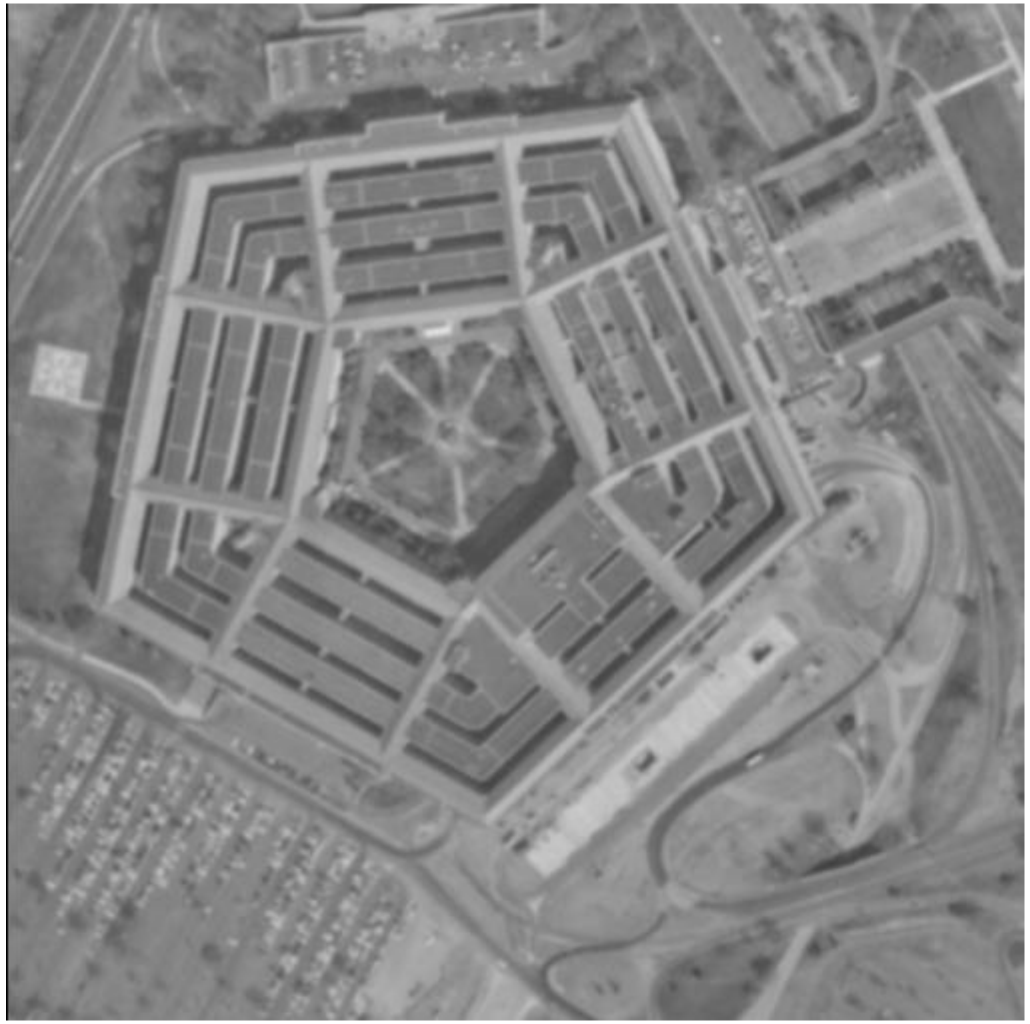
가우시안 블러링 소스코드



```
void Gausian_blur(){
    int i,j,k,l;
    float sum;

    float gaussian[5][5] = {{0.003765, 0.015019, 0.023792, 0.015019, 0.003765},
                            {0.015019, 0.059912, 0.094907, 0.059912, 0.015019},
                            {0.023792, 0.094907, 0.150342, 0.094907, 0.023792},
                            {0.015019, 0.059912, 0.094907, 0.059912, 0.015019},
                            {0.003765, 0.015019, 0.023792, 0.015019, 0.003765}};

    for(i=2; i<h; i++){
        for(j=2; j<w; j++){
            sum=0;
            for(k=0; k<5; k++){
                for(l=0; l<5; l++){
                    sum+=(float)IpImg[(i*width)+(k*width)-(2*width)+j-2+l]*gaussian[k][l];
                }
            }
            BlurImg[i*width+j]=(BYTE)(sum+0.5);
        }
    }
    return;
}
```

가우시안 블러링 후 이미지





소벨 엣지 검출

대상 이미지

-1	0	1
-2	0	2
-1	0	1

수평 마스크

$\longrightarrow$ Gx

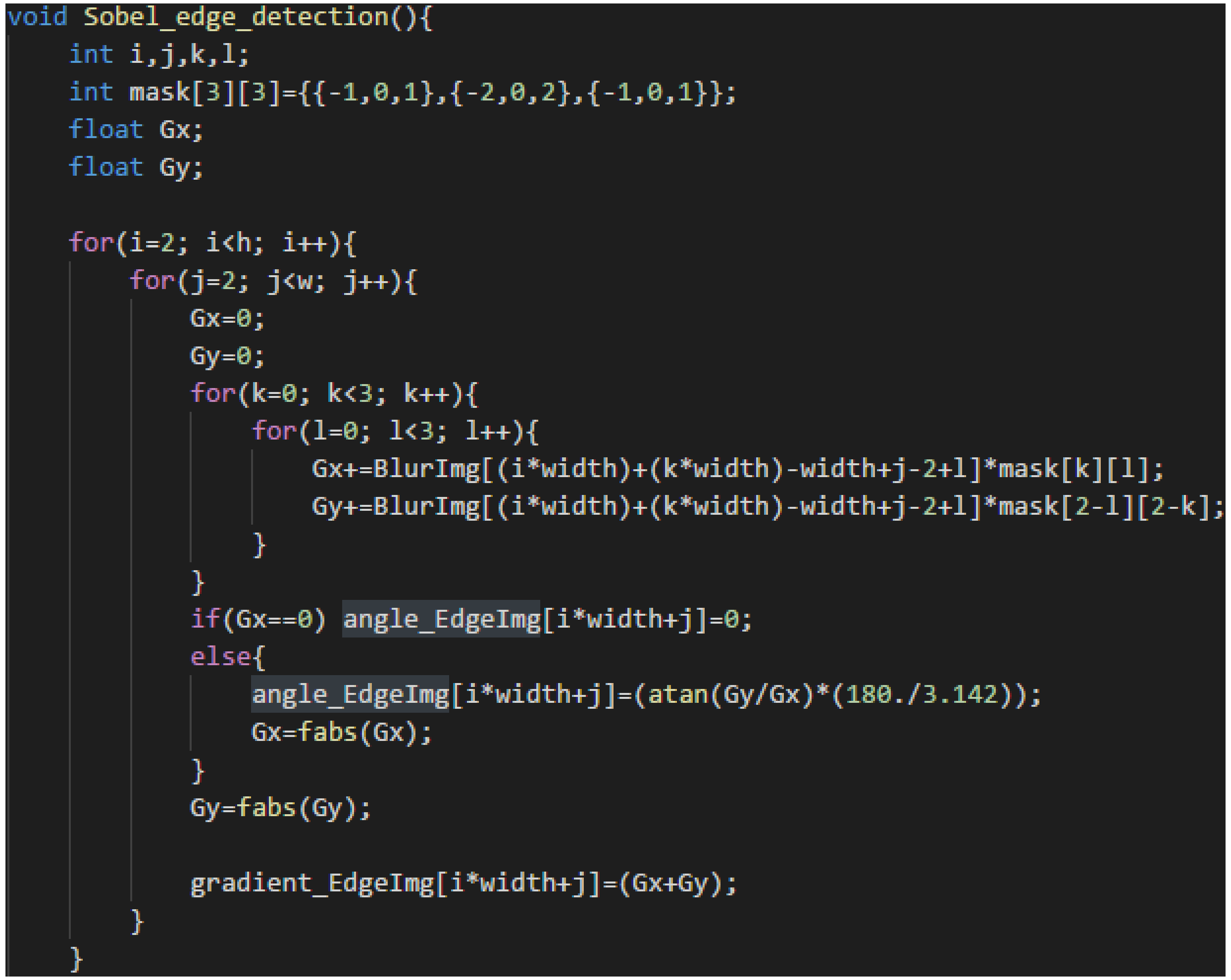
1	2	1
0	0	0
-1	-2	1

수직 마스크

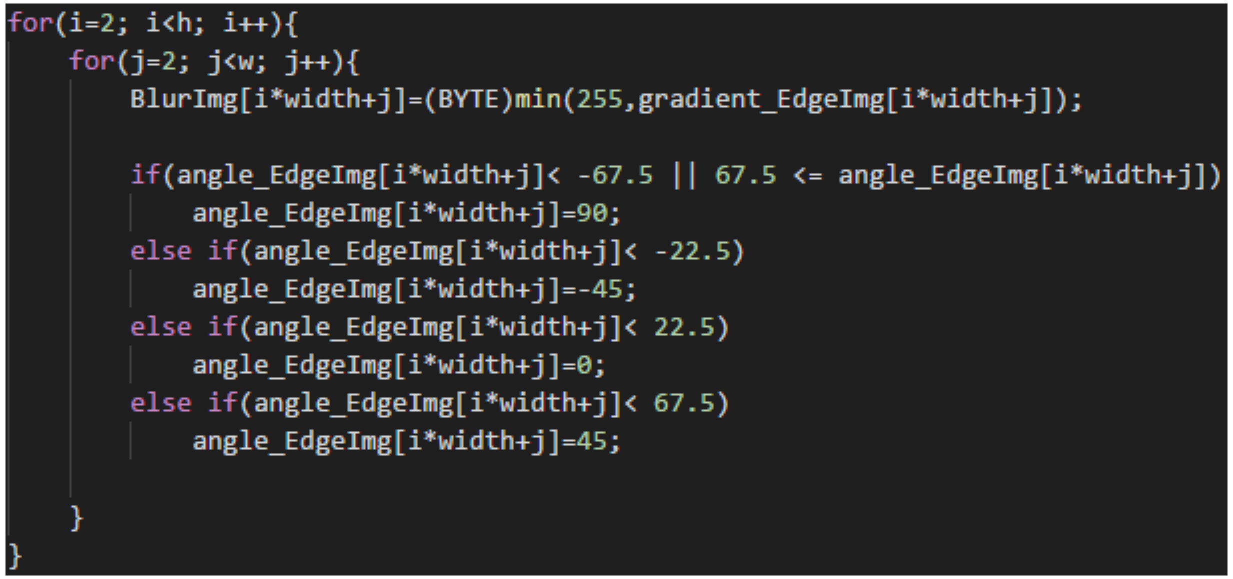
$\longrightarrow$ Gy

magnitude=|Gx|+|Gy|

소벨 엣지 검출 소스코드

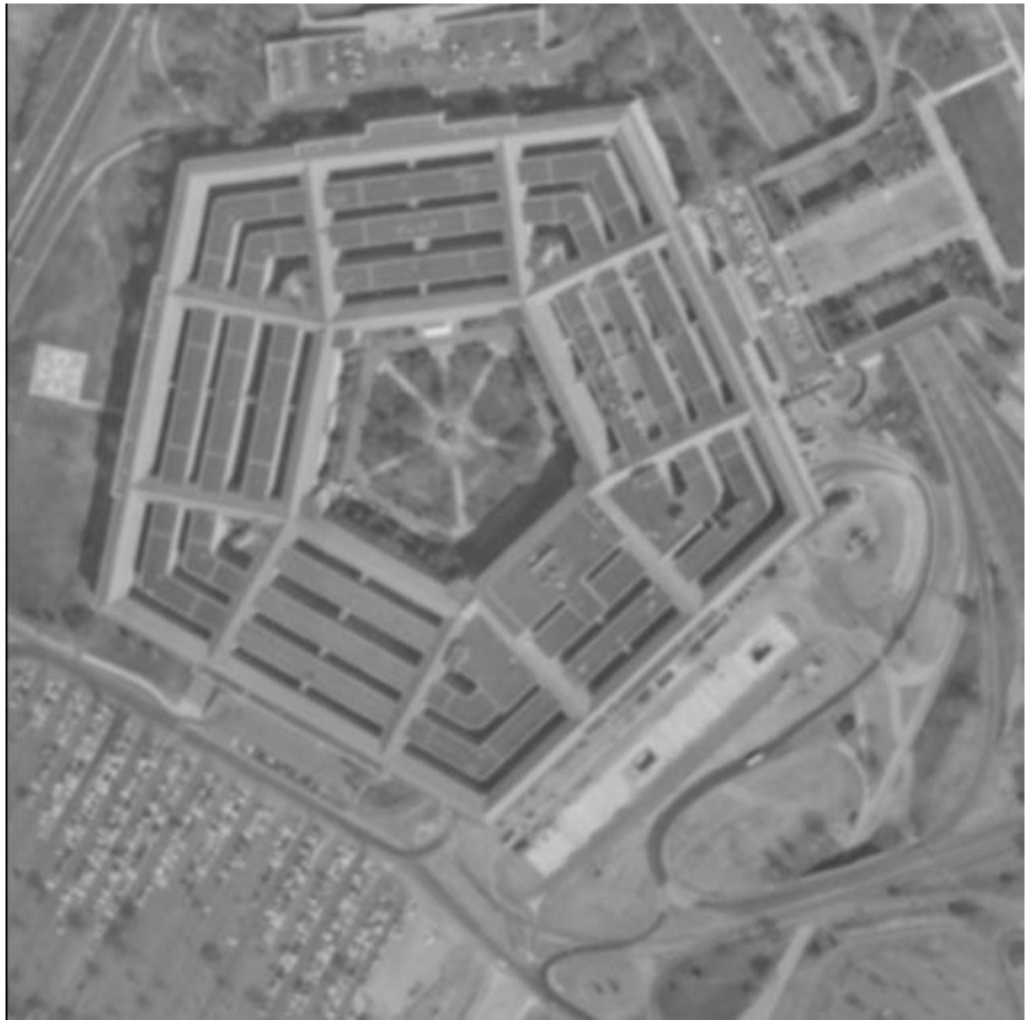


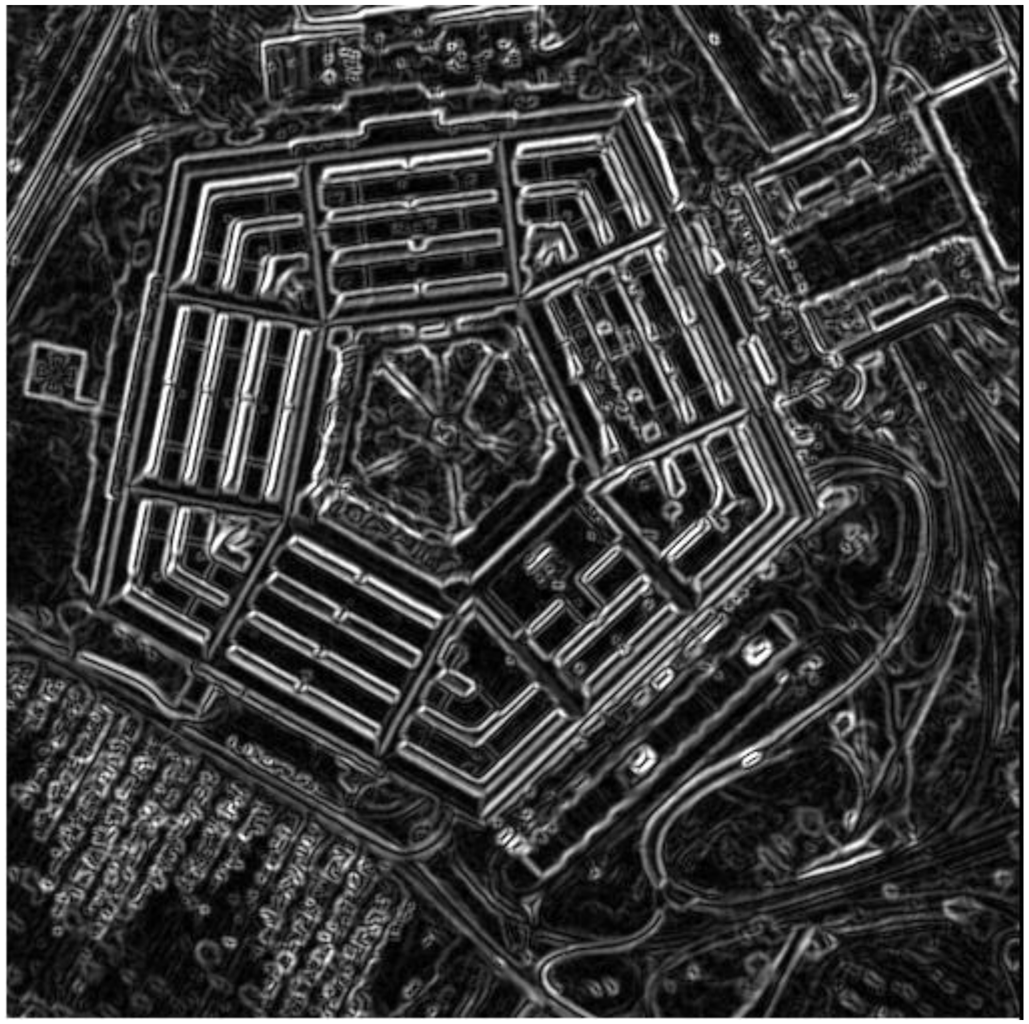
```
void Sobel_edge_detection(){
    int i,j,k,l;
    int mask[3][3]={{-1,0,1},{-2,0,2},{-1,0,1}};
    float Gx;
    float Gy;

    for(i=2; i<h; i++){
        for(j=2; j<w; j++){
            Gx=0;
            Gy=0;
            for(k=0; k<3; k++){
                for(l=0; l<3; l++){
                    Gx+=BlurImg[(i*width)+(k*width)-width+j-2+l]*mask[k][l];
                    Gy+=BlurImg[(i*width)+(k*width)-width+j-2+l]*mask[2-l][2-k];
                }
            }
            if(Gx==0) angle_EdgeImg[i*width+j]=0;
            else{
                angle_EdgeImg[i*width+j]=(atan(Gy/Gx)*(180./3.142));
                Gx=fabs(Gx);
            }
            Gy=fabs(Gy);

            gradient_EdgeImg[i*width+j]=(Gx+Gy);
        }
    }
```



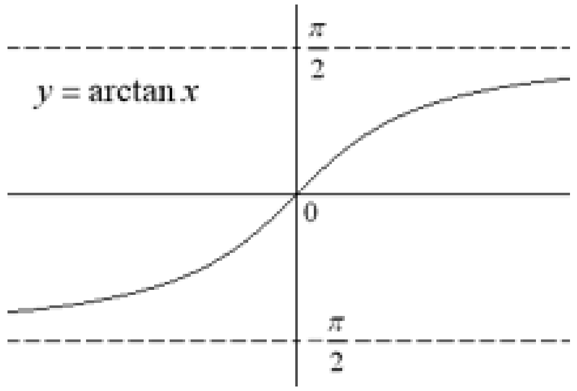
```
for(i=2; i<h; i++){
    for(j=2; j<w; j++){
        BlurImg[i*width+j]=(BYTE)min(255,gradient_EdgeImg[i*width+j]);

        if(angle_EdgeImg[i*width+j]< -67.5 || 67.5 <= angle_EdgeImg[i*width+j])
            angle_EdgeImg[i*width+j]=90;
        else if(angle_EdgeImg[i*width+j]< -22.5)
            angle_EdgeImg[i*width+j]=-45;
        else if(angle_EdgeImg[i*width+j]< 22.5)
            angle_EdgeImg[i*width+j]=0;
        else if(angle_EdgeImg[i*width+j]< 67.5)
            angle_EdgeImg[i*width+j]=45;

    }
}
```

엣지 검출 후 이미지





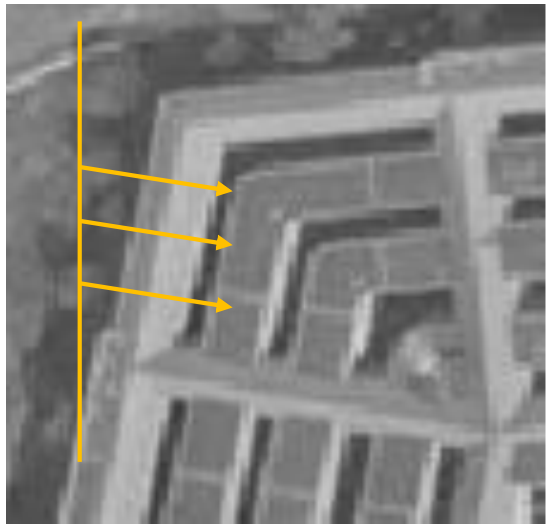
비최대치 억제



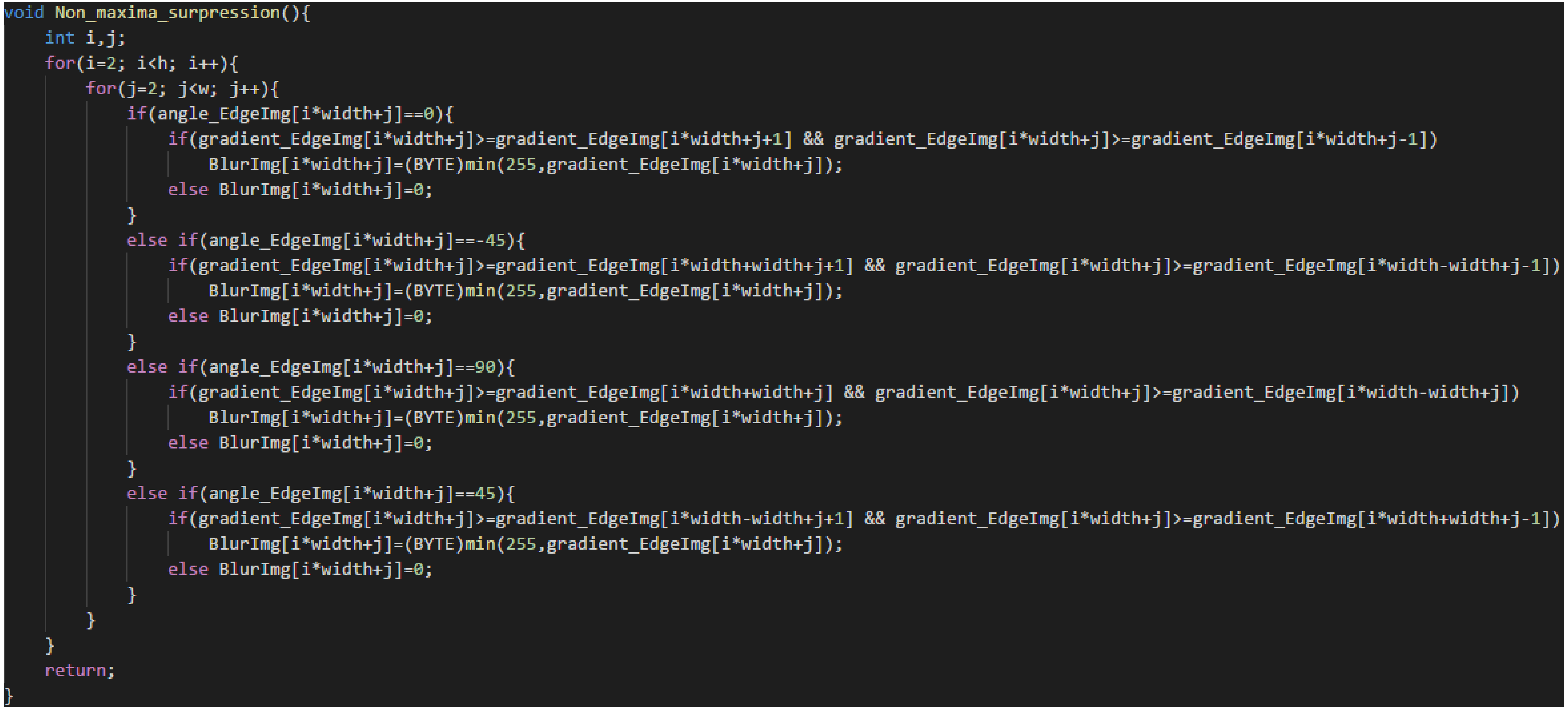
$$\text{angle of gradient} = \arctan \frac{Gy}{Gx}$$

$$\text{angle of gradient} + \frac{\pi}{2} = \text{angle of edge}$$

0	0	Theta
0	Theta	0
Theta	0	0

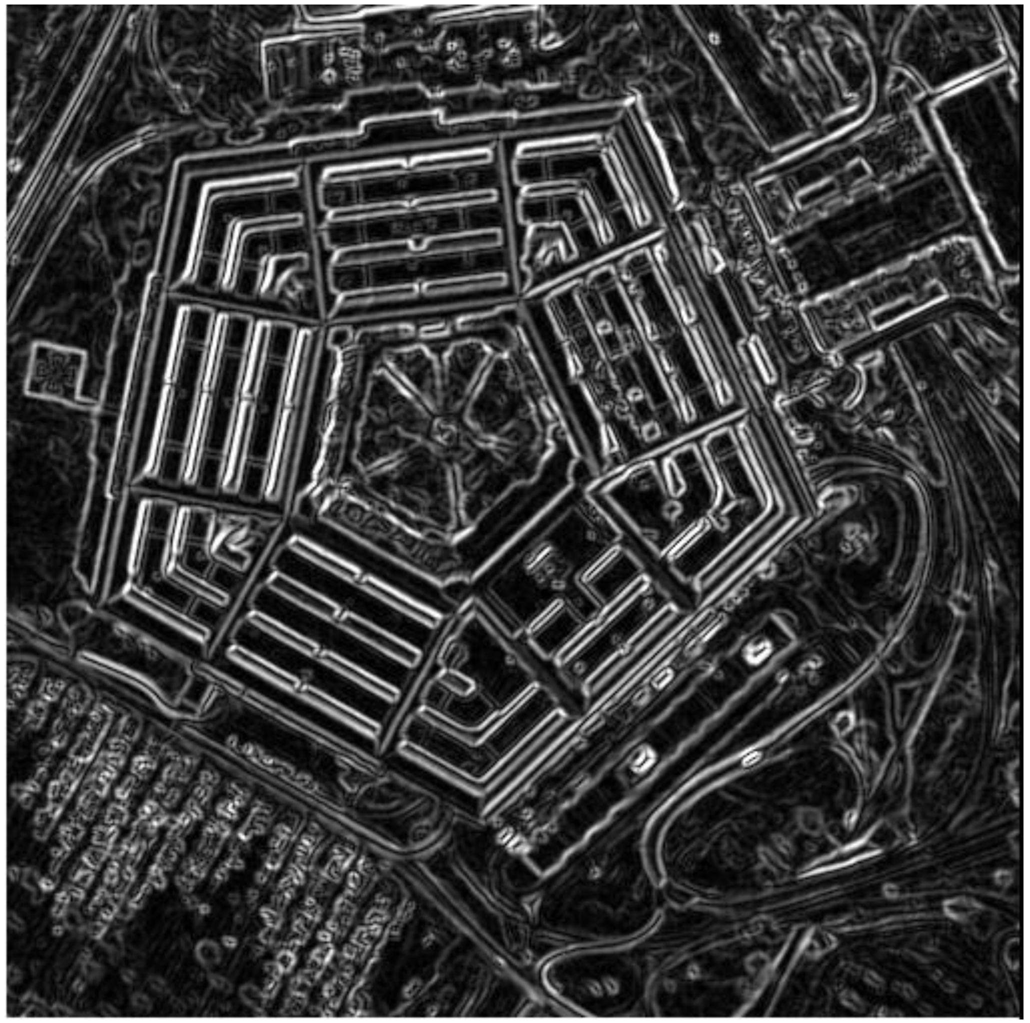


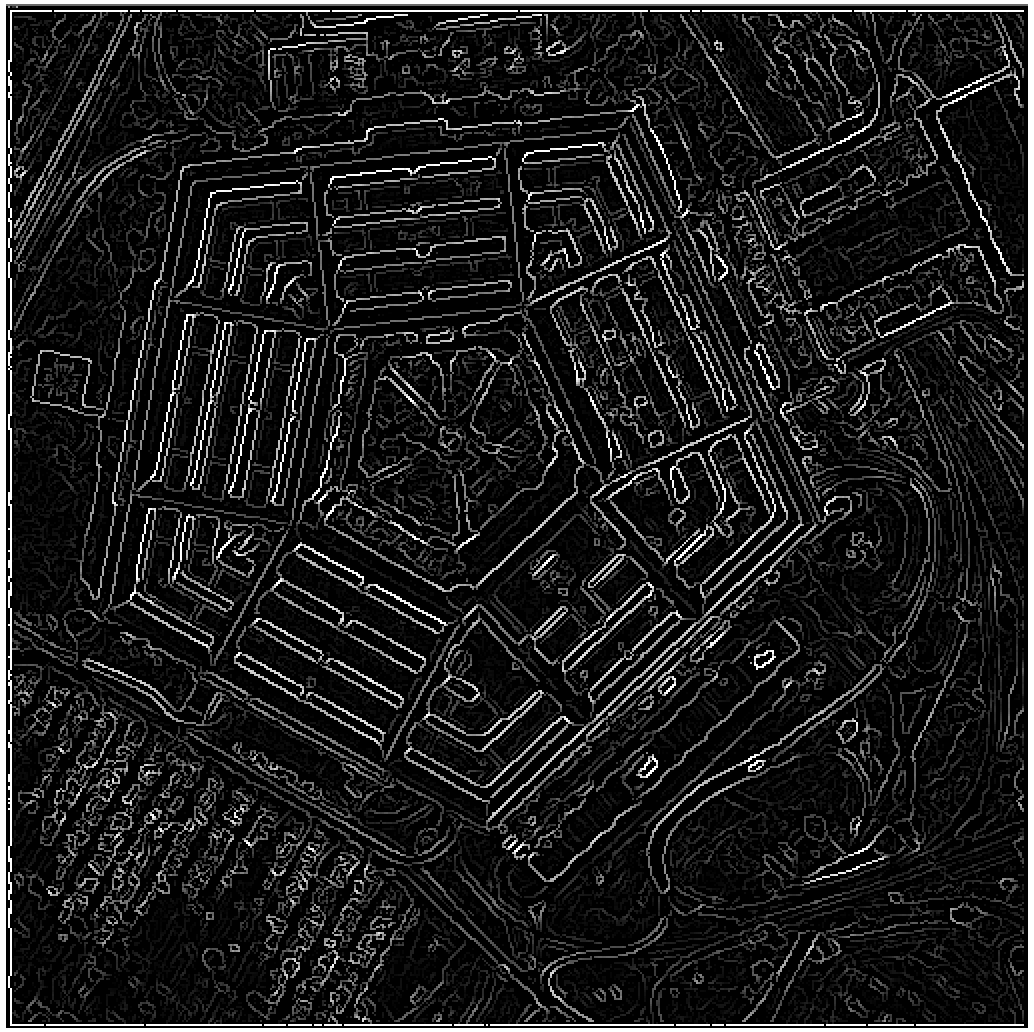
비최대치 억제 소스코드



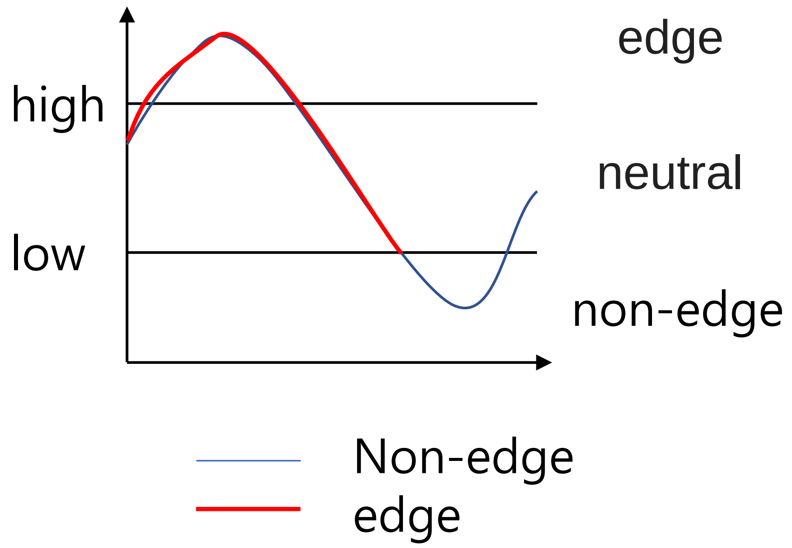
```
void Non_maxima_surpression(){
    int i,j;
    for(i=2; i<h; i++){
        for(j=2; j<w; j++){
            if(angle_EdgeImg[i*width+j]==0){
                if(gradient_EdgeImg[i*width+j]>=gradient_EdgeImg[i*width+j+1] && gradient_EdgeImg[i*width+j]>=gradient_EdgeImg[i*width+j-1])
                    BlurImg[i*width+j]=(BYTE)min(255,gradient_EdgeImg[i*width+j]);
                else BlurImg[i*width+j]=0;
            }
            else if(angle_EdgeImg[i*width+j]==-45){
                if(gradient_EdgeImg[i*width+j]>=gradient_EdgeImg[i*width+width+j+1] && gradient_EdgeImg[i*width+j]>=gradient_EdgeImg[i*width-width+j-1])
                    BlurImg[i*width+j]=(BYTE)min(255,gradient_EdgeImg[i*width+j]);
                else BlurImg[i*width+j]=0;
            }
            else if(angle_EdgeImg[i*width+j]==90){
                if(gradient_EdgeImg[i*width+j]>=gradient_EdgeImg[i*width+width+j] && gradient_EdgeImg[i*width+j]>=gradient_EdgeImg[i*width-width+j])
                    BlurImg[i*width+j]=(BYTE)min(255,gradient_EdgeImg[i*width+j]);
                else BlurImg[i*width+j]=0;
            }
            else if(angle_EdgeImg[i*width+j]==45){
                if(gradient_EdgeImg[i*width+j]>=gradient_EdgeImg[i*width-width+j+1] && gradient_EdgeImg[i*width+j]>=gradient_EdgeImg[i*width+width+j-1])
                    BlurImg[i*width+j]=(BYTE)min(255,gradient_EdgeImg[i*width+j]);
                else BlurImg[i*width+j]=0;
            }
        }
    }
    return;
}
```

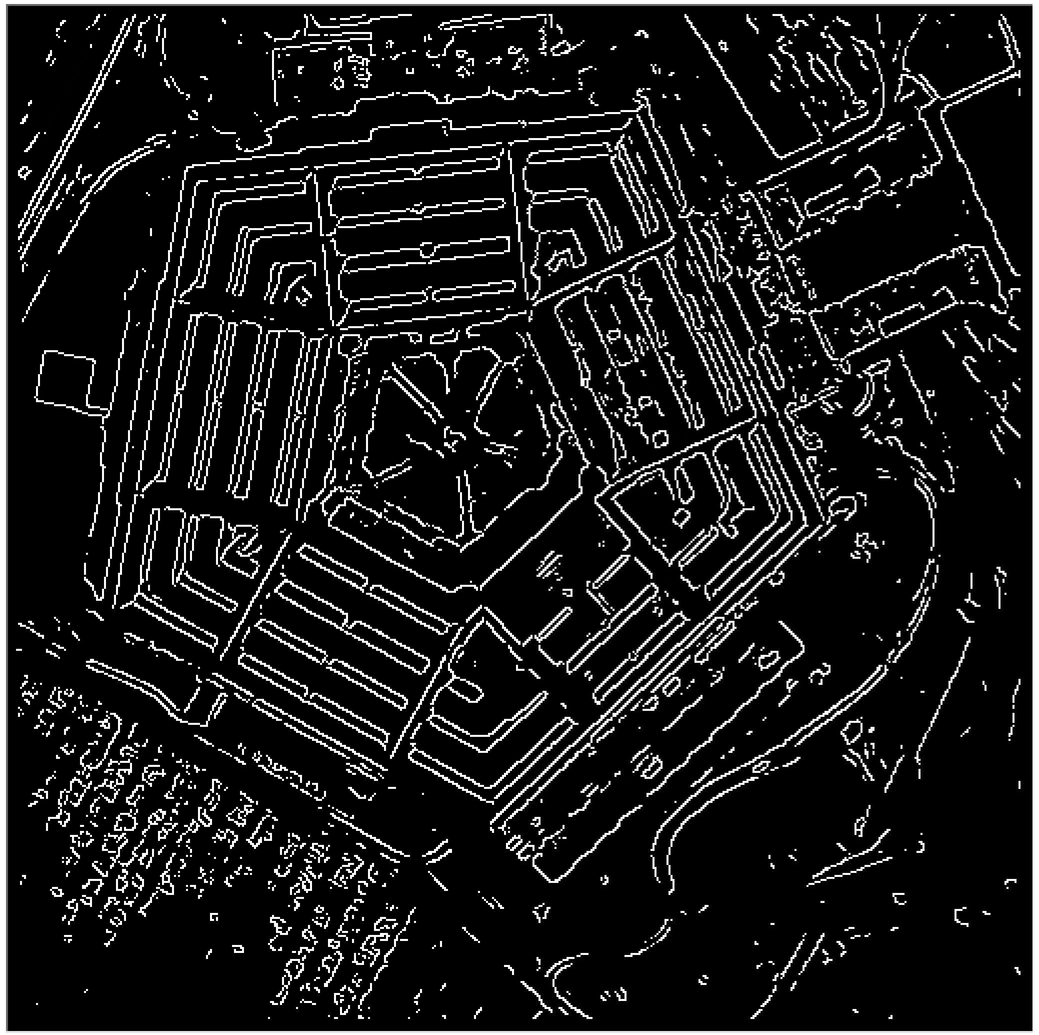
엣지 thinning 후 이미지



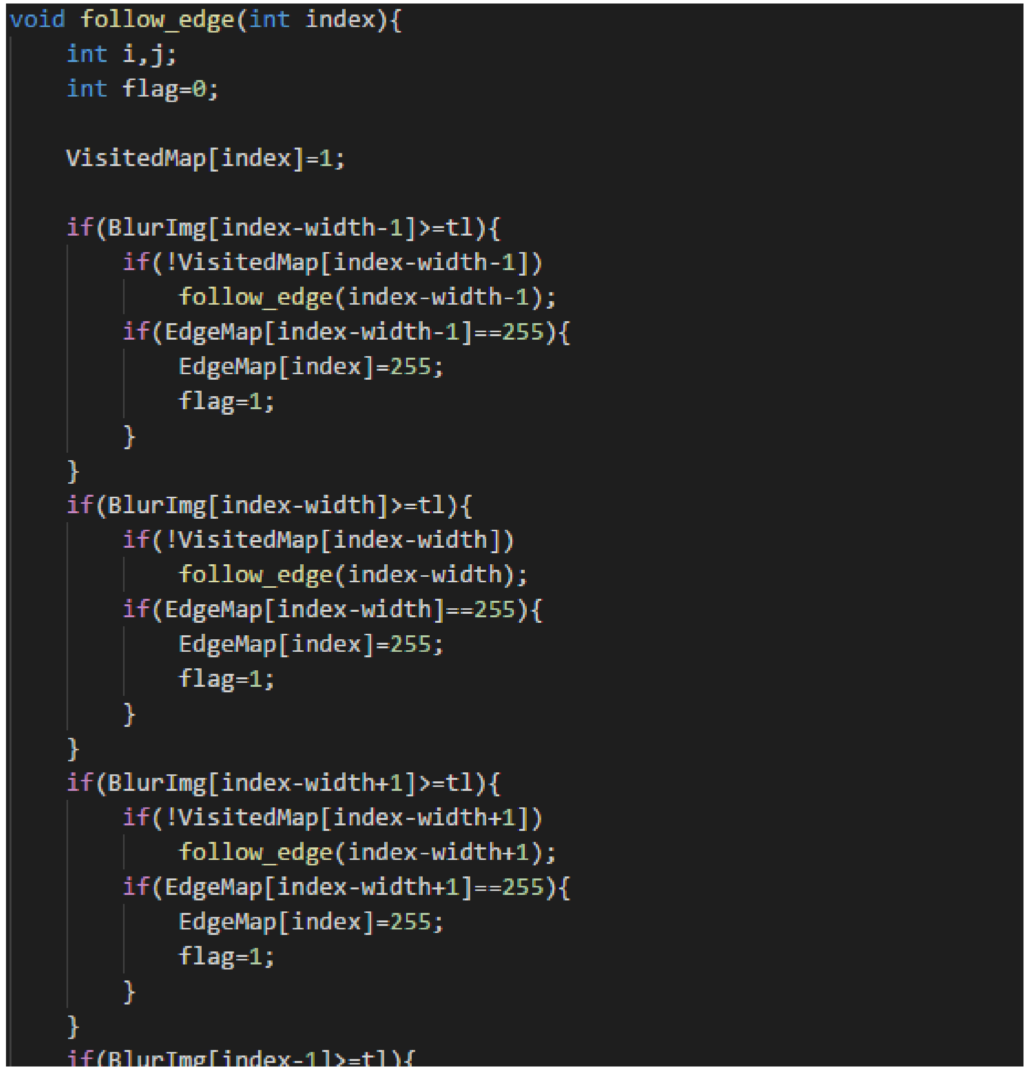


히스테리시스 스레스홀딩

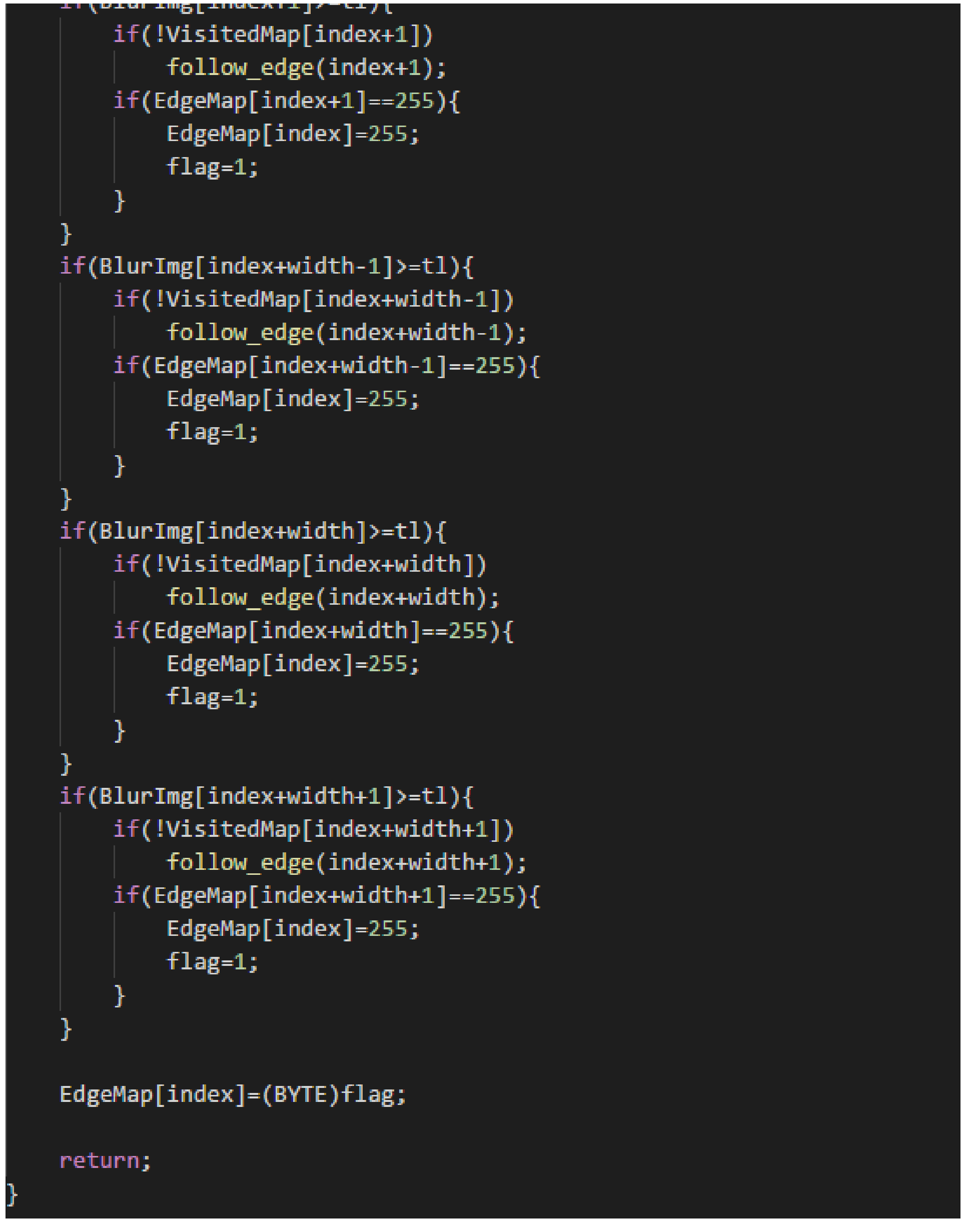




히스테리시스 스레스홀딩 소스코드

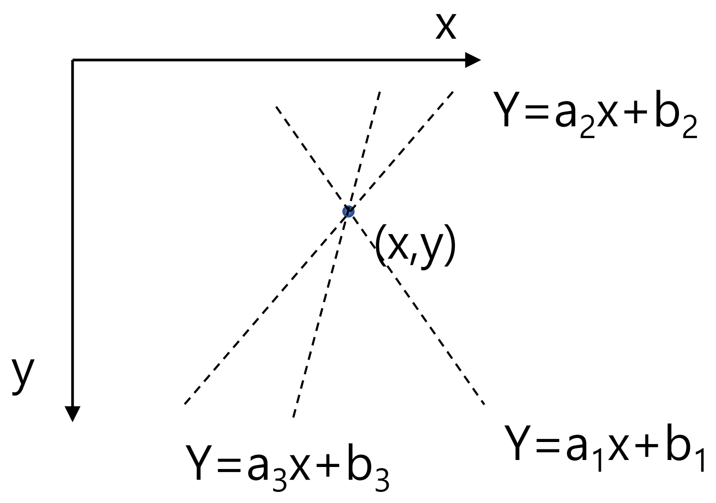


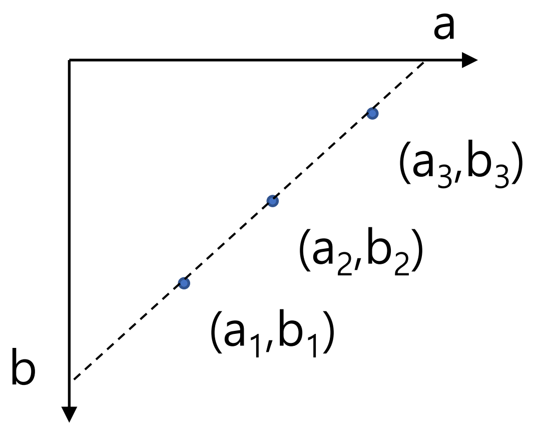
```
void follow_edge(int index){
    int i,j;
    int flag=0;

    VisitedMap[index]=1;

    if(BlurImg[index-width-1]>=tl){
        if(!VisitedMap[index-width-1])
            follow_edge(index-width-1);
        if(EdgeMap[index-width-1]==255){
            EdgeMap[index]=255;
            flag=1;
        }
    }
    if(BlurImg[index-width]>=tl){
        if(!VisitedMap[index-width])
            follow_edge(index-width);
        if(EdgeMap[index-width]==255){
            EdgeMap[index]=255;
            flag=1;
        }
    }
    if(BlurImg[index-width+1]>=tl){
        if(!VisitedMap[index-width+1])
            follow_edge(index-width+1);
        if(EdgeMap[index-width+1]==255){
            EdgeMap[index]=255;
            flag=1;
        }
    }
    if(BlurImg[index-1]>=tl){
```



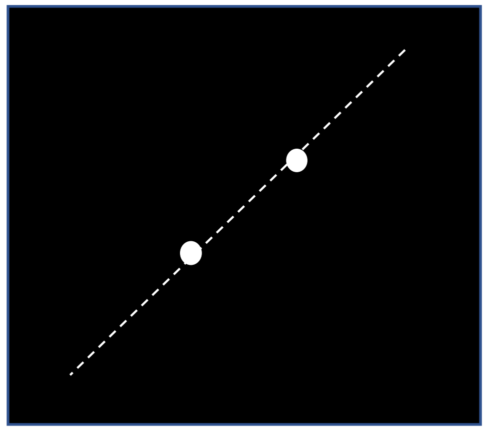
```
    if(BlurImg[index+1]>=tl){
        if(!VisitedMap[index+1])
            follow_edge(index+1);
        if(EdgeMap[index+1]==255){
            EdgeMap[index]=255;
            flag=1;
        }
    }
    if(BlurImg[index+width-1]>=tl){
        if(!VisitedMap[index+width-1])
            follow_edge(index+width-1);
        if(EdgeMap[index+width-1]==255){
            EdgeMap[index]=255;
            flag=1;
        }
    }
    if(BlurImg[index+width]>=tl){
        if(!VisitedMap[index+width])
            follow_edge(index+width);
        if(EdgeMap[index+width]==255){
            EdgeMap[index]=255;
            flag=1;
        }
    }
    if(BlurImg[index+width+1]>=tl){
        if(!VisitedMap[index+width+1])
            follow_edge(index+width+1);
        if(EdgeMap[index+width+1]==255){
            EdgeMap[index]=255;
            flag=1;
        }
    }

    EdgeMap[index]=(BYTE)flag;

    return;
}
```

허프 변환

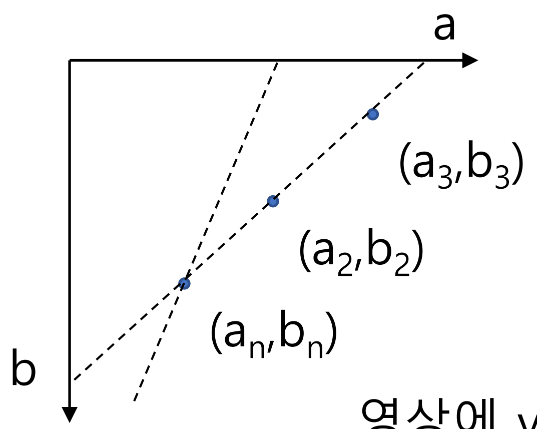




$b=-ax+y$

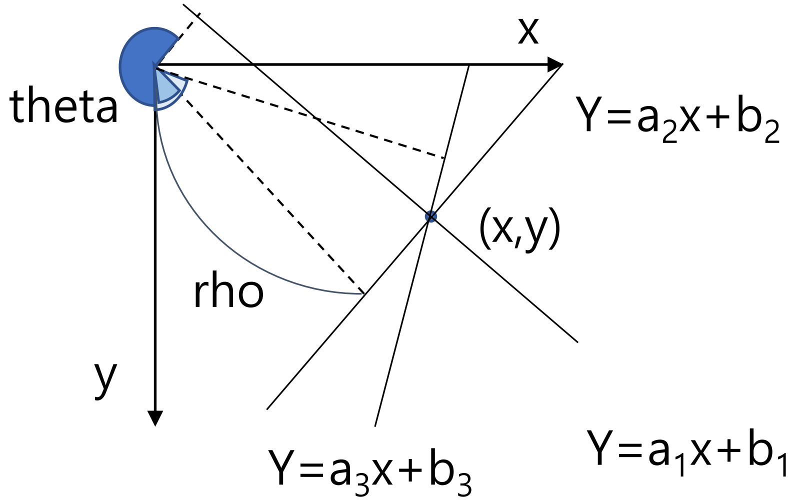


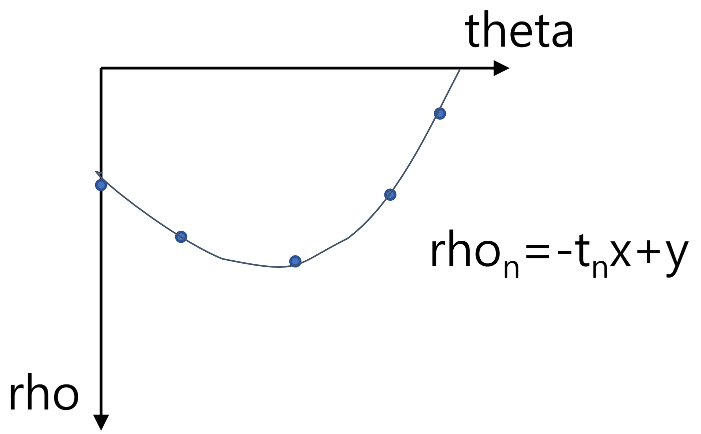
$Y=a_nx+b_n$



영상에 $y= a_nx+b_n$인
직선이 존재할 가능성

허프 변환





rho :원점으로 부터 직선에 그린 수직선의 길이

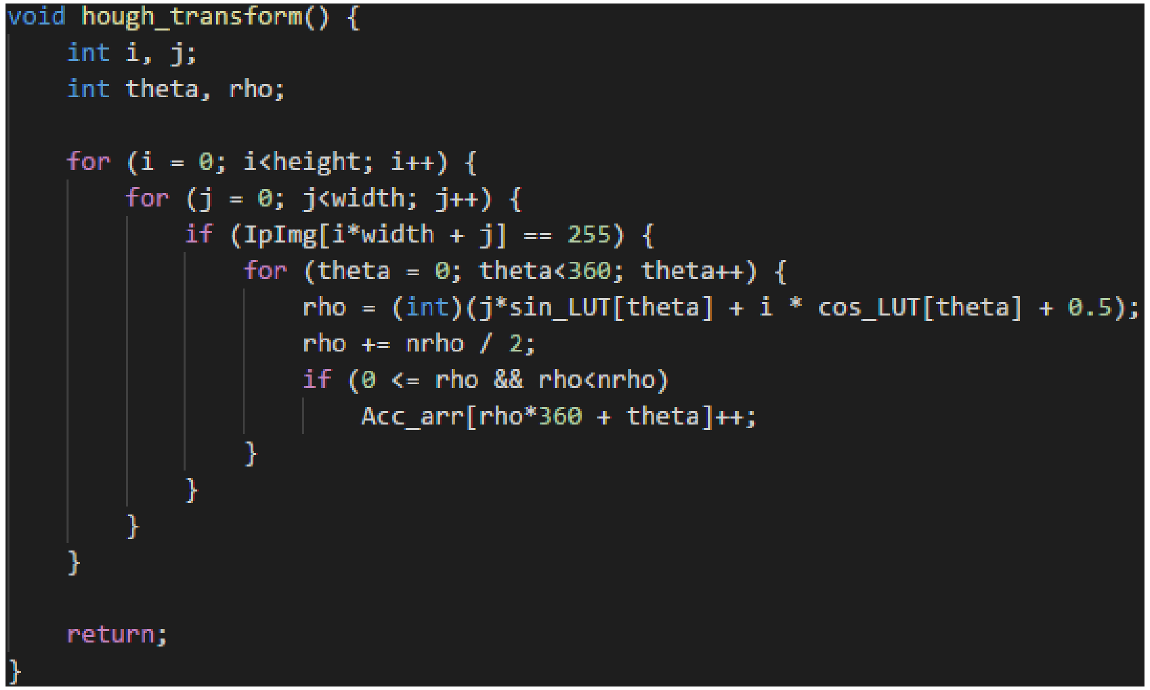
theta :수직선과 y축이 만드는 각 (0~ 2π)

$rho = \sin\Theta \, x + \cos\Theta \, y$

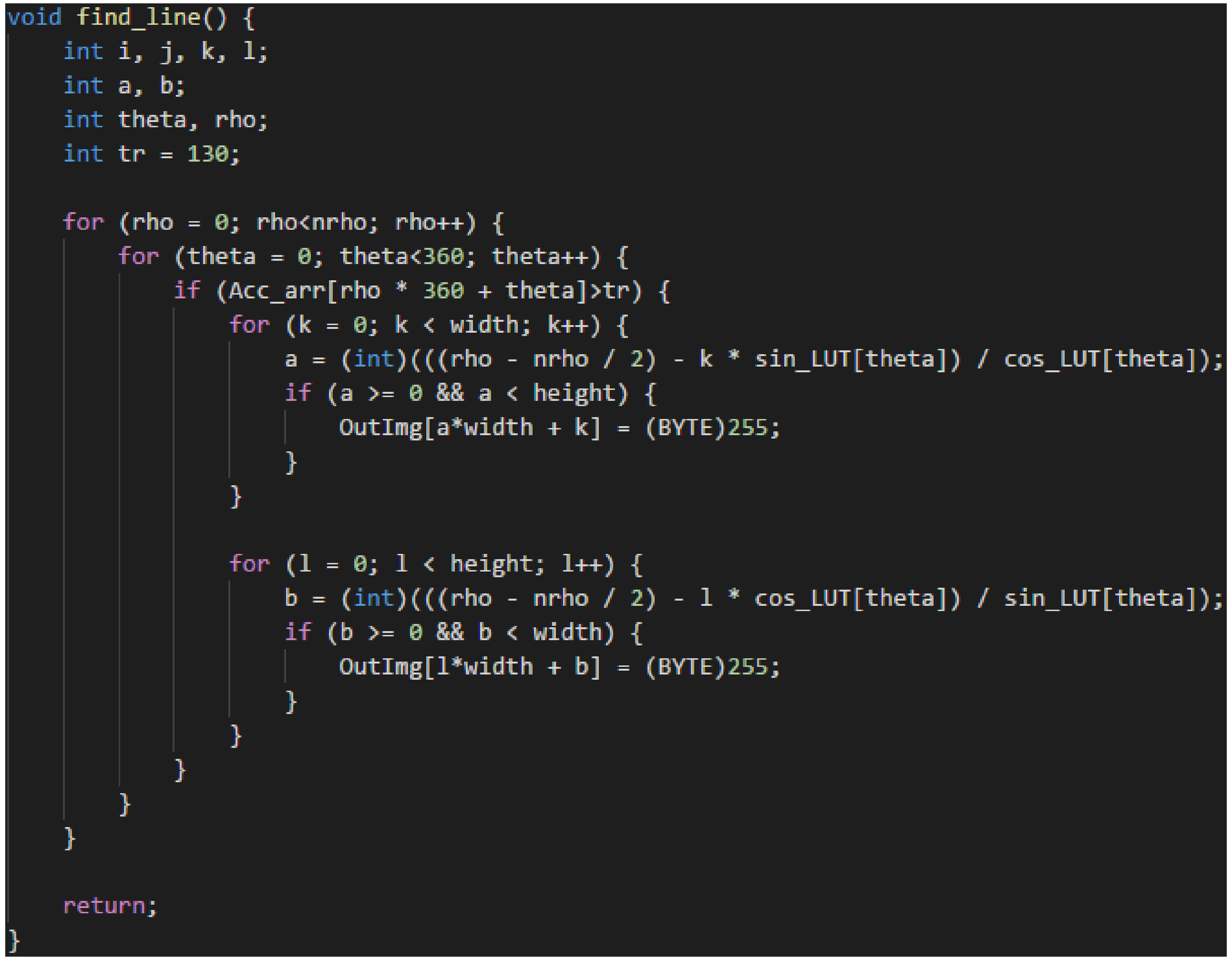
$x = (rho - \cos\Theta \, y) / \sin\Theta$

$y = (rho - \sin\Theta \, x) / \cos\Theta$

허프 변환 소스코드



```c
void hough_transform() {
    int i, j;
    int theta, rho;

    for (i = 0; i<height; i++) {
        for (j = 0; j<width; j++) {
            if (IpImg[i*width + j] == 255) {
                for (theta = 0; theta<360; theta++) {
                    rho = (int)(j*sin_LUT[theta] + i * cos_LUT[theta] + 0.5);
                    rho += nrho / 2;
                    if (0 <= rho && rho<nrho)
                        Acc_arr[rho*360 + theta]++;
                }
            }
        }
    }

    return;
}
```



```c
void find_line() {
    int i, j, k, l;
    int a, b;
    int theta, rho;
    int tr = 130;

    for (rho = 0; rho<nrho; rho++) {
        for (theta = 0; theta<360; theta++) {
            if (Acc_arr[rho * 360 + theta]>tr) {
                for (k = 0; k < width; k++) {
                    a = (int)(((rho - nrho / 2) - k * sin_LUT[theta]) / cos_LUT[theta]);
                    if (a >= 0 && a < height) {
                        OutImg[a*width + k] = (BYTE)255;
                    }
                }

                for (l = 0; l < height; l++) {
                    b = (int)(((rho - nrho / 2) - l * cos_LUT[theta]) / sin_LUT[theta]);
                    if (b >= 0 && b < width) {
                        OutImg[l*width + b] = (BYTE)255;
                    }
                }
            }
        }
    }

    return;
}
```

결과 이미지