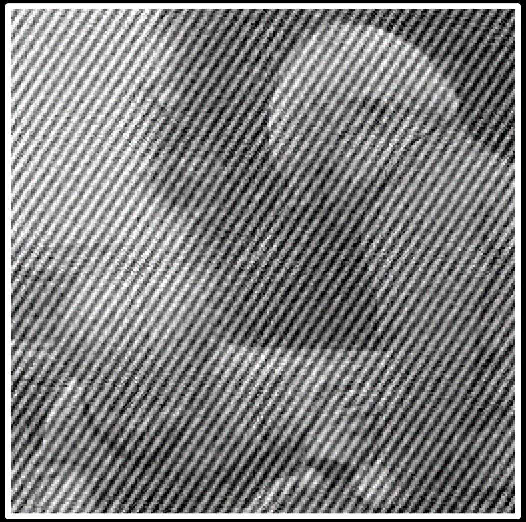
DSP실험 프로젝트

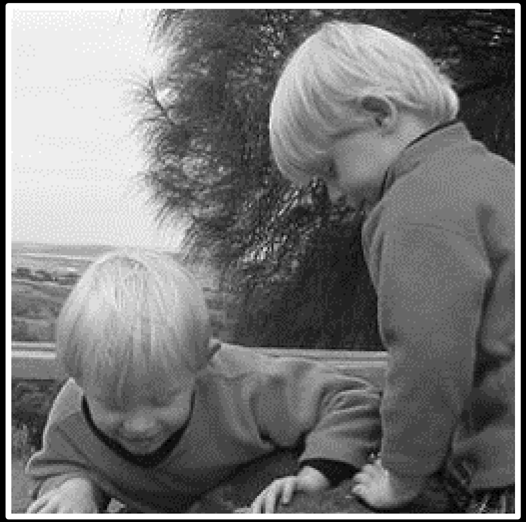
조장 : 이효건

조원 : 안유민 유지나 이종혁 정수현

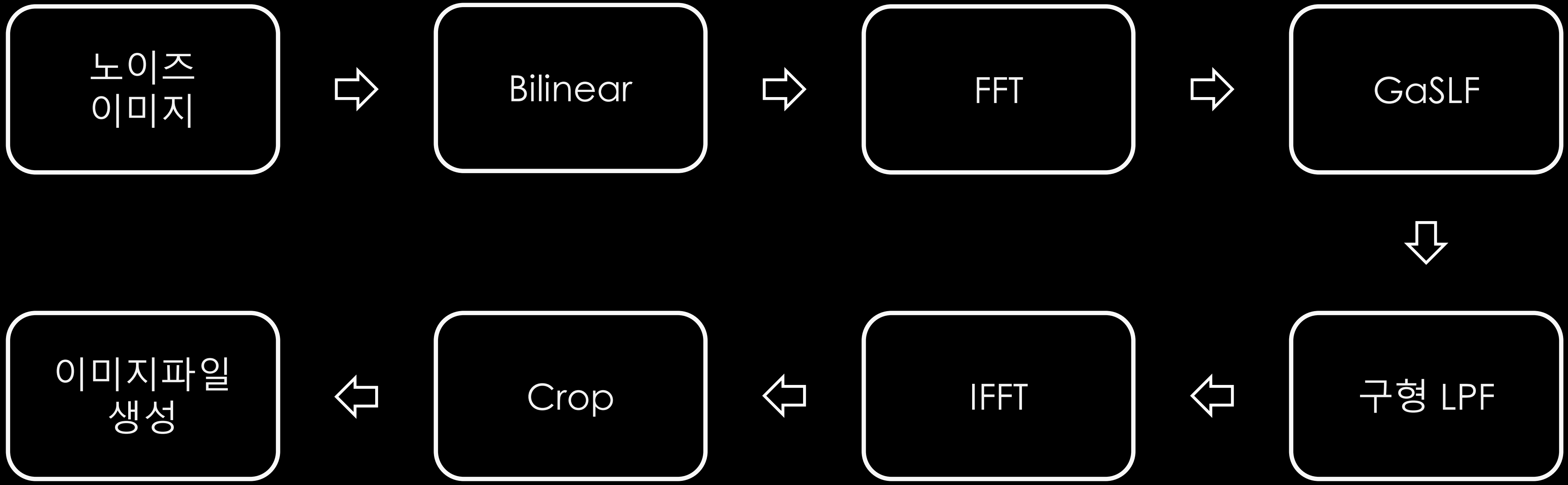
0. 목차

1. 필터링 목표

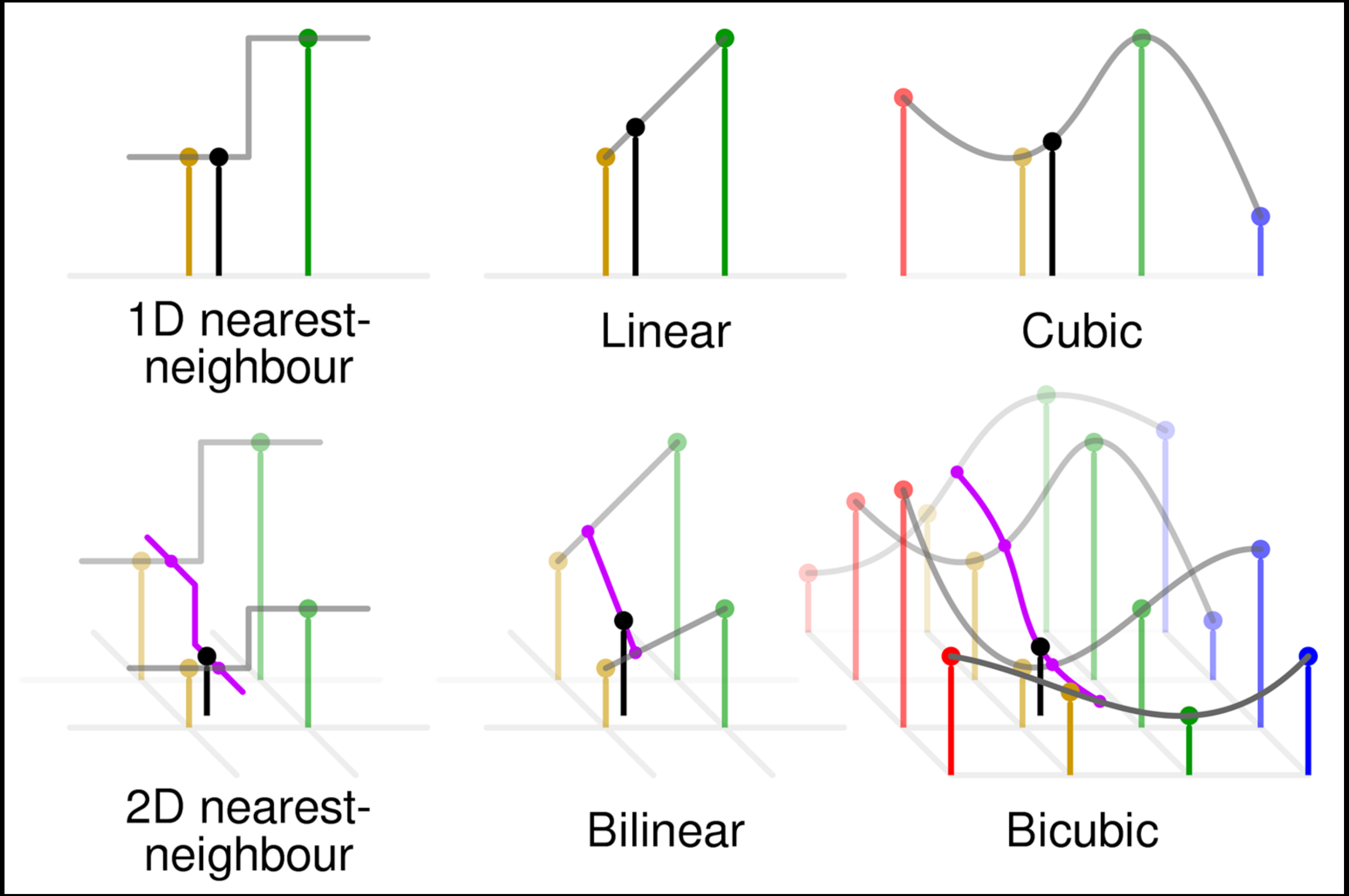




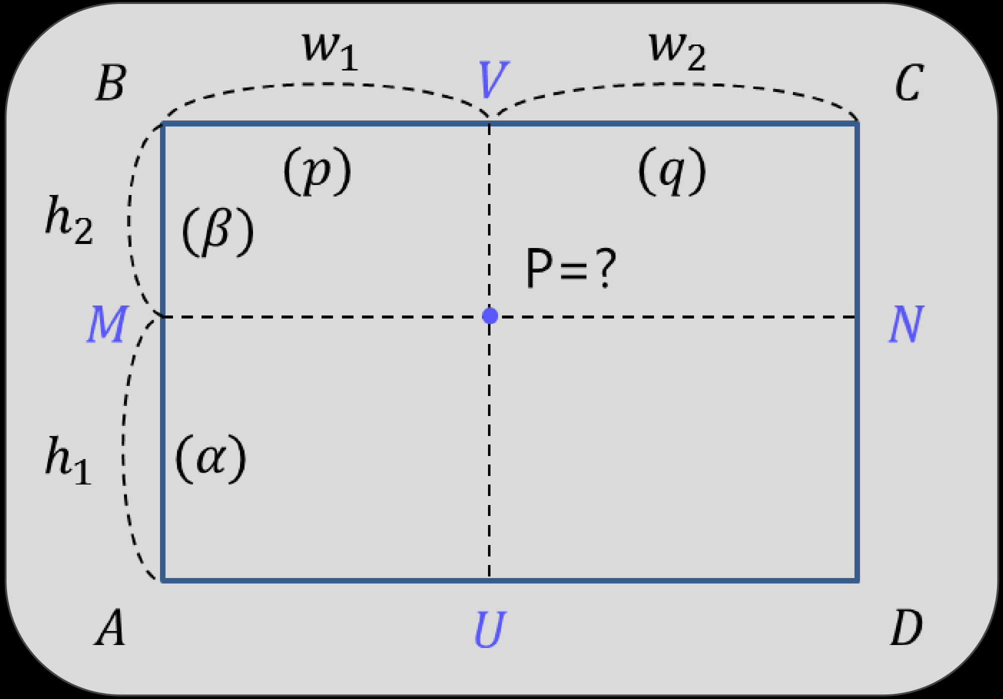
2. 알고리즘



BILINEAR 함수

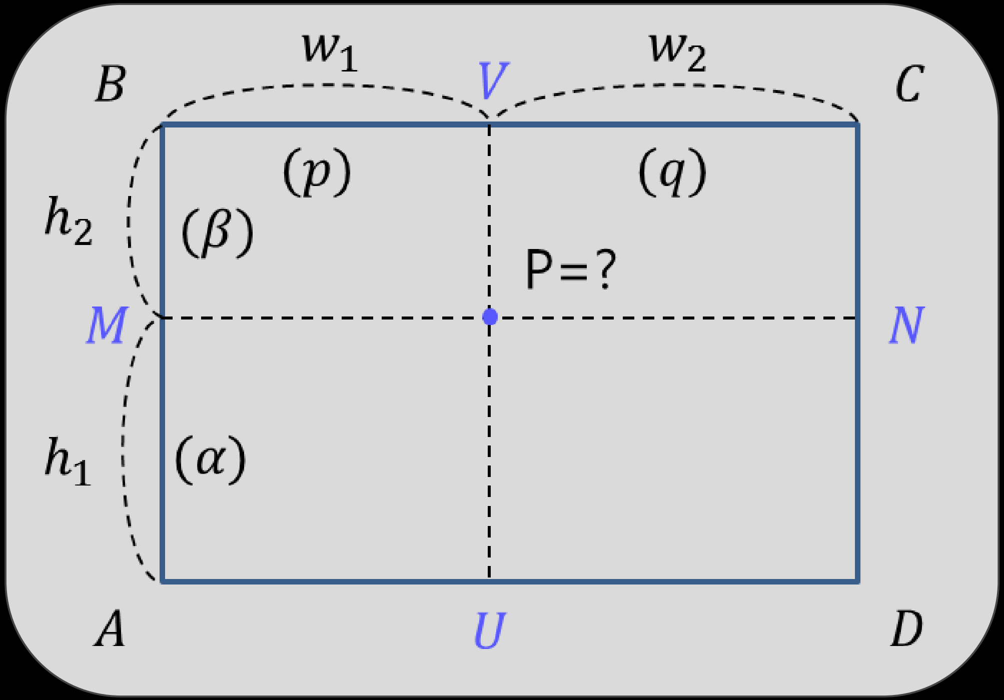


Wikipedia : Bilinear Interpolation



다크프로그래머 블로그

BILINEAR 함수



다크프로그래머 블로그

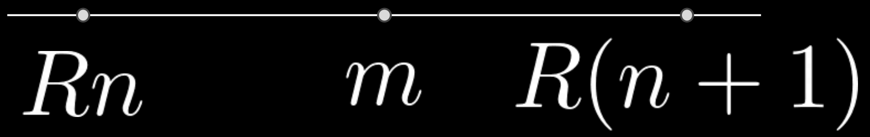
$$\alpha = \frac{h1}{h1+h2} \quad \beta = \frac{h2}{h1+h2}$$

$$p = \frac{w1}{w1+w2} \quad q = \frac{w2}{w1+w2}$$

$$\begin{aligned} P &= q(\beta A + \alpha B) + p(\beta D + \alpha C) \\ &= q\beta A + q\alpha B + p\beta D + p\alpha C \end{aligned}$$

BILINEAR 함수

R : Rate



$$\alpha = \frac{h1}{h1+h2} \quad \beta = \frac{h2}{h1+h2}$$

$$p = \frac{w1}{w1+w2} \quad q = \frac{w2}{w1+w2}$$

$$\begin{aligned} P &= q(\beta A + \alpha B) + p(\beta D + \alpha C) \\ &= q\beta A + q\alpha B + p\beta D + p\alpha C \end{aligned}$$

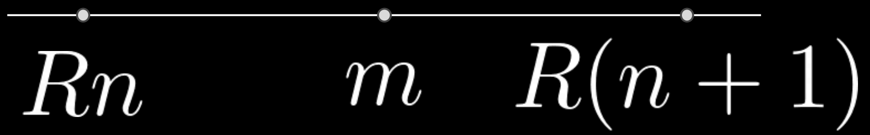
BILINEAR 함수

$$\alpha = \frac{h1}{h1+h2} \quad \beta = \frac{h2}{h1+h2}$$

$$p = \frac{w1}{w1+w2} \quad q = \frac{w2}{w1+w2}$$

$$\begin{aligned} P &= q(\beta A + \alpha B) + p(\beta D + \alpha C) \\ &= q\beta A + q\alpha B + p\beta D + p\alpha C \end{aligned}$$

R : Rate



$$Rn \le m \le R(n+1)$$

$$n \le \frac{m}{R} \le n+1$$

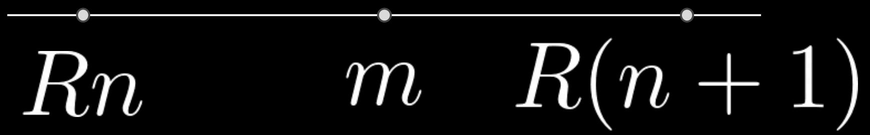
BILINEAR 함수

$$\alpha = \frac{h1}{h1 + h2} \quad \beta = \frac{h2}{h1 + h2}$$

$$p = \frac{w1}{w1 + w2} \quad q = \frac{w2}{w1 + w2}$$

$$\begin{aligned} P &= q(\beta A + \alpha B) + p(\beta D + \alpha C) \\ &= q\beta A + q\alpha B + p\beta D + p\alpha C \end{aligned}$$

R : Rate

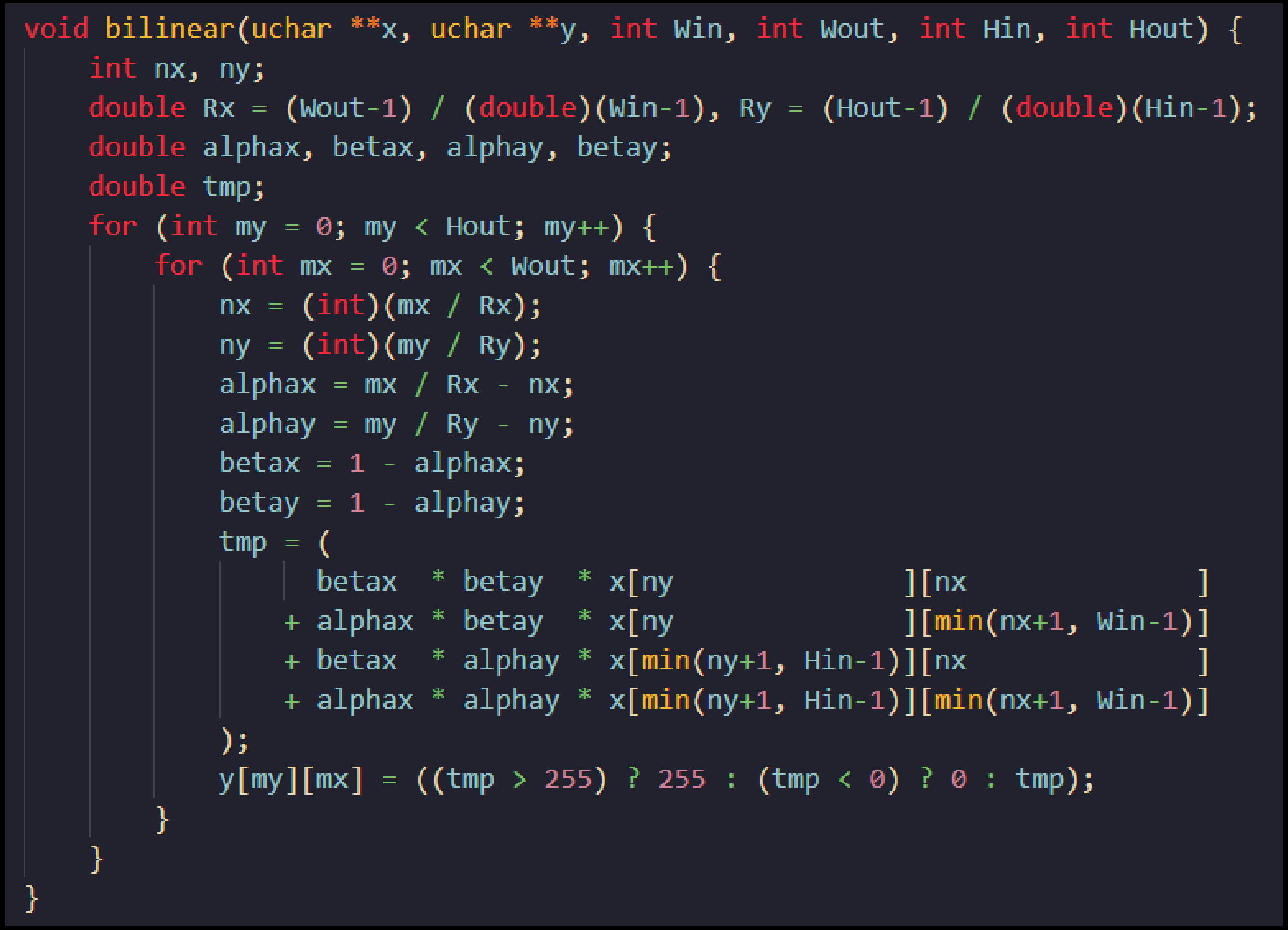


$$Rn \le m \le R(n+1)$$

$$n \le \frac{m}{R} \le n + 1$$

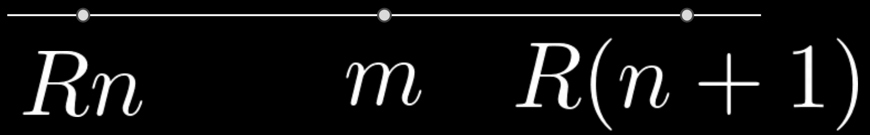
$$\therefore n = \left\lfloor \frac{m}{R} \right\rfloor$$

BILINEAR 함수



```c
void bilinear(uchar **x, uchar **y, int Win, int Wout, int Hin, int Hout) {
    int nx, ny;
    double Rx = (Wout-1) / (double)(Win-1), Ry = (Hout-1) / (double)(Hin-1);
    double alphax, betax, alphay, betay;
    double tmp;
    for (int my = 0; my < Hout; my++) {
        for (int mx = 0; mx < Wout; mx++) {
            nx = (int)(mx / Rx);
            ny = (int)(my / Ry);
            alphax = mx / Rx - nx;
            alphay = my / Ry - ny;
            betax = 1 - alphax;
            betay = 1 - alphay;
            tmp = (
                  betax  * betay  * x[ny              ][nx               ]
                + alphax * betay  * x[ny              ][min(nx+1, Win-1)]
                + betax  * alphay * x[min(ny+1, Hin-1)][nx               ]
                + alphax * alphay * x[min(ny+1, Hin-1)][min(nx+1, Win-1)]
            );
            y[my][mx] = ((tmp > 255) ? 255 : (tmp < 0) ? 0 : tmp);
        }
    }
}
```

R : Rate



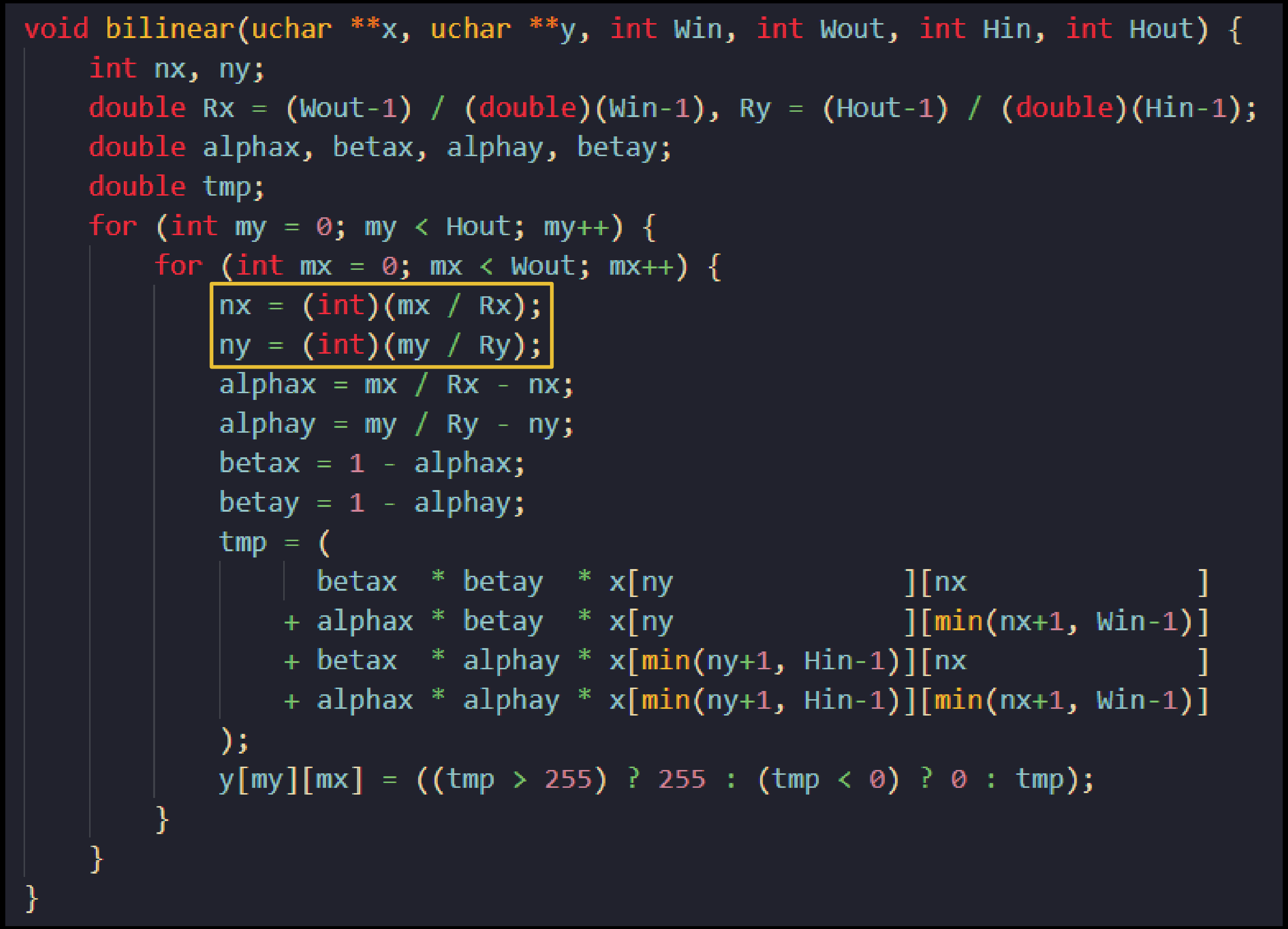
$$Rn \leq m \leq R(n+1)$$

$$n \leq \frac{m}{R} \leq n+1$$

$$\therefore n = \left\lfloor \frac{m}{R} \right\rfloor$$

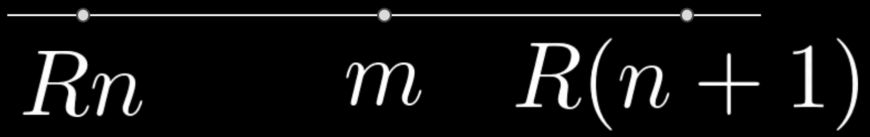
알고리즘

BILINEAR 함수



```
void bilinear(uchar **x, uchar **y, int Win, int Wout, int Hin, int Hout) {
    int nx, ny;
    double Rx = (Wout-1) / (double)(Win-1), Ry = (Hout-1) / (double)(Hin-1);
    double alphax, betax, alphay, betay;
    double tmp;
    for (int my = 0; my < Hout; my++) {
        for (int mx = 0; mx < Wout; mx++) {
            nx = (int)(mx / Rx);
            ny = (int)(my / Ry);
            alphax = mx / Rx - nx;
            alphay = my / Ry - ny;
            betax = 1 - alphax;
            betay = 1 - alphay;
            tmp = (
                  betax  * betay  * x[ny              ][nx              ]
                + alphax * betay  * x[ny              ][min(nx+1, Win-1)]
                + betax  * alphay * x[min(ny+1, Hin-1)][nx              ]
                + alphax * alphay * x[min(ny+1, Hin-1)][min(nx+1, Win-1)]
            );
            y[my][mx] = ((tmp > 255) ? 255 : (tmp < 0) ? 0 : tmp);
        }
    }
}
```

R : Rate

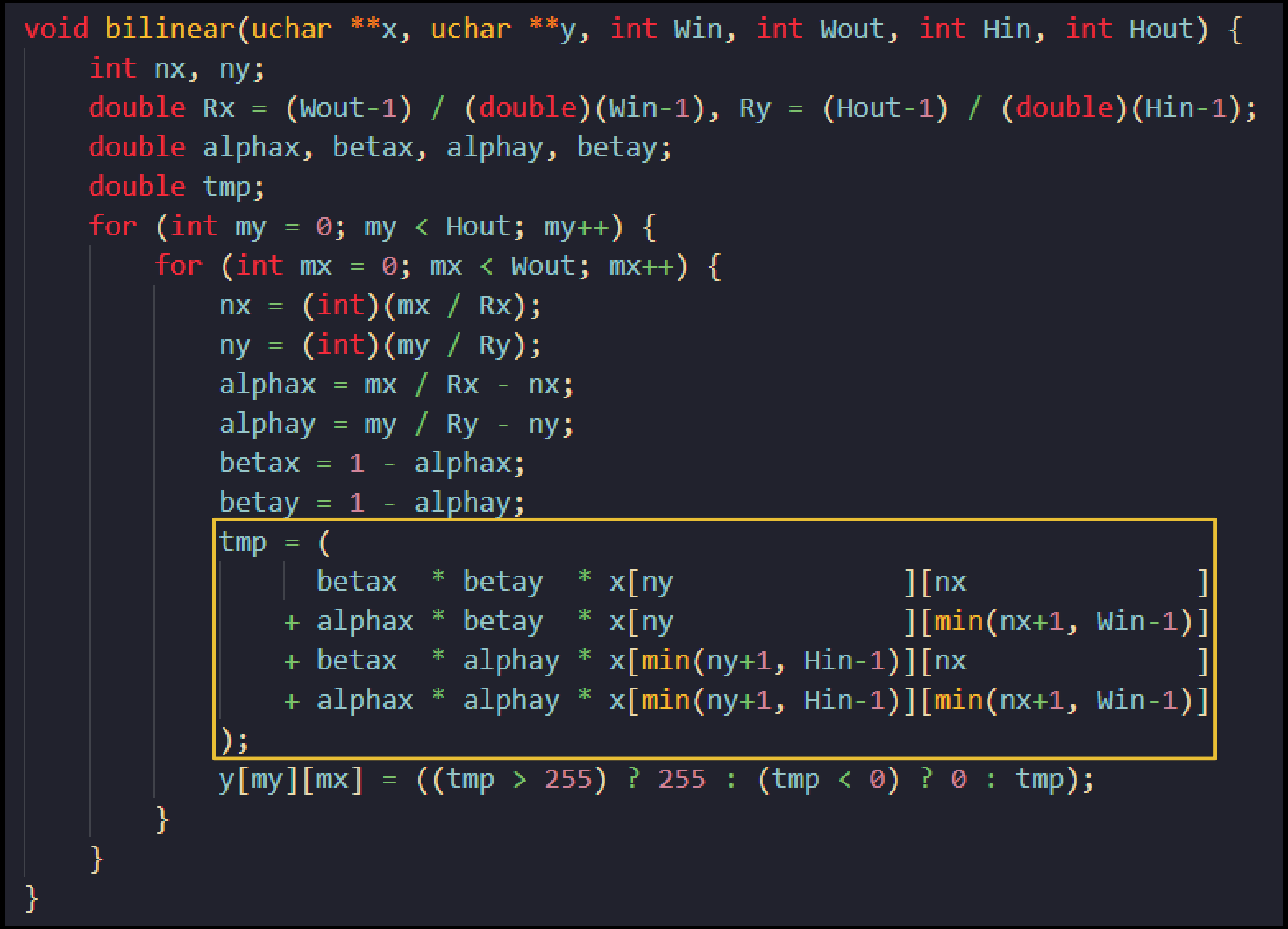


$$Rn \leq m \leq R(n+1)$$

$$n \leq \frac{m}{R} \leq n+1$$

$$\therefore n = \left\lfloor \frac{m}{R} \right\rfloor$$

BILINEAR 함수



```
void bilinear(uchar **x, uchar **y, int Win, int Wout, int Hin, int Hout) {
    int nx, ny;
    double Rx = (Wout-1) / (double)(Win-1), Ry = (Hout-1) / (double)(Hin-1);
    double alphax, betax, alphay, betay;
    double tmp;
    for (int my = 0; my < Hout; my++) {
        for (int mx = 0; mx < Wout; mx++) {
            nx = (int)(mx / Rx);
            ny = (int)(my / Ry);
            alphax = mx / Rx - nx;
            alphay = my / Ry - ny;
            betax = 1 - alphax;
            betay = 1 - alphay;
            tmp = (
                  betax  * betay  * x[ny                ][nx                ]
                + alphax * betay  * x[ny                ][min(nx+1, Win-1)]
                + betax  * alphay * x[min(ny+1, Hin-1)][nx                ]
                + alphax * alphay * x[min(ny+1, Hin-1)][min(nx+1, Win-1)]
            );
            y[my][mx] = ((tmp > 255) ? 255 : (tmp < 0) ? 0 : tmp);
        }
    }
}
```

$$\alpha = \frac{h1}{h1 + h2} \quad \beta = \frac{h2}{h1 + h2}$$

$$p = \frac{w1}{w1 + w2} \quad q = \frac{w2}{w1 + w2}$$

$$\begin{aligned} P &= q(\beta A + \alpha B) + p(\beta D + \alpha C) \\ &= q\beta A + q\alpha B + p\beta D + p\alpha C \end{aligned}$$

BILINEAR 함수

BILINEAR 함수

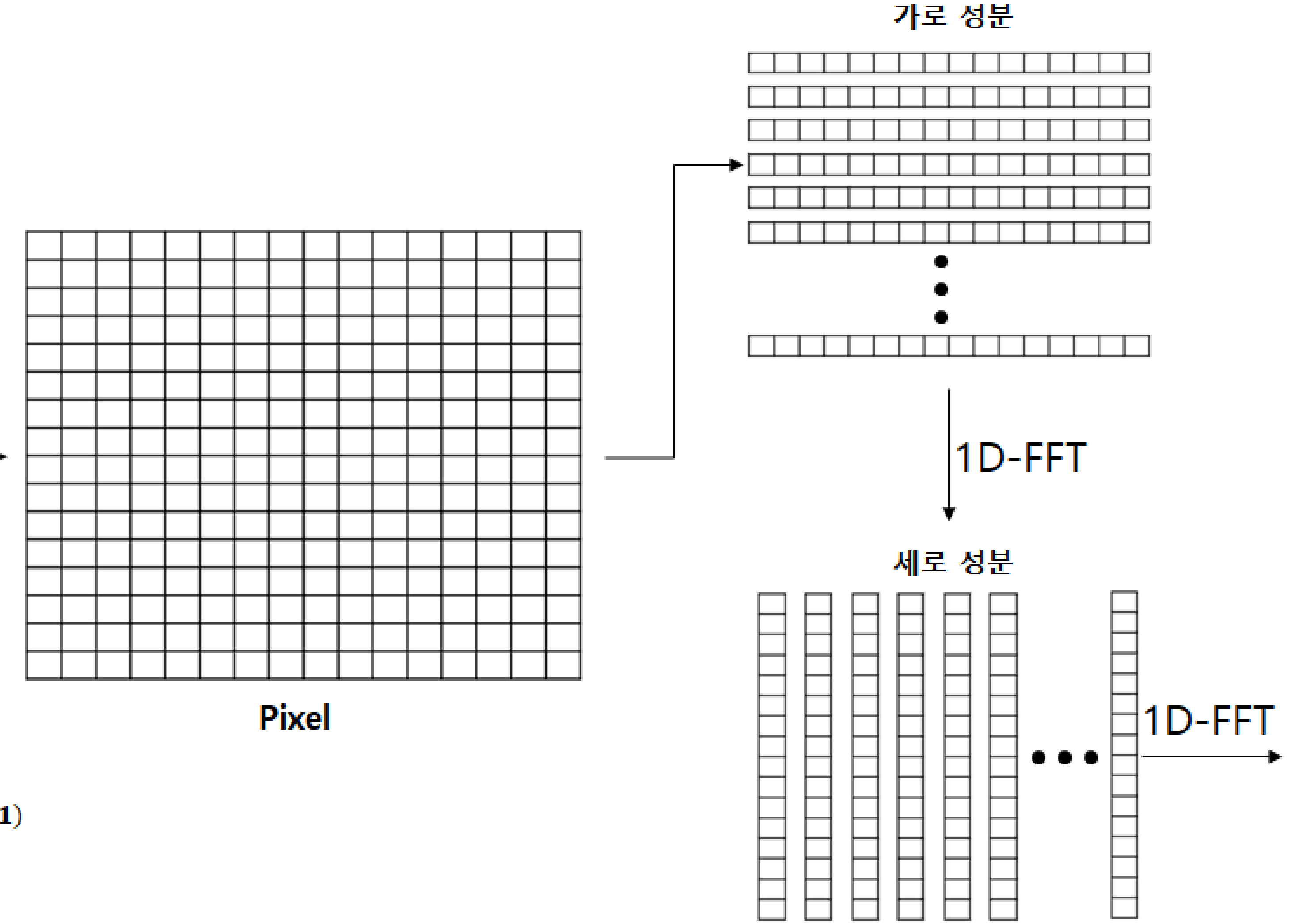
노이즈 원본 이미지

크기가 변경된 이미지

FFT

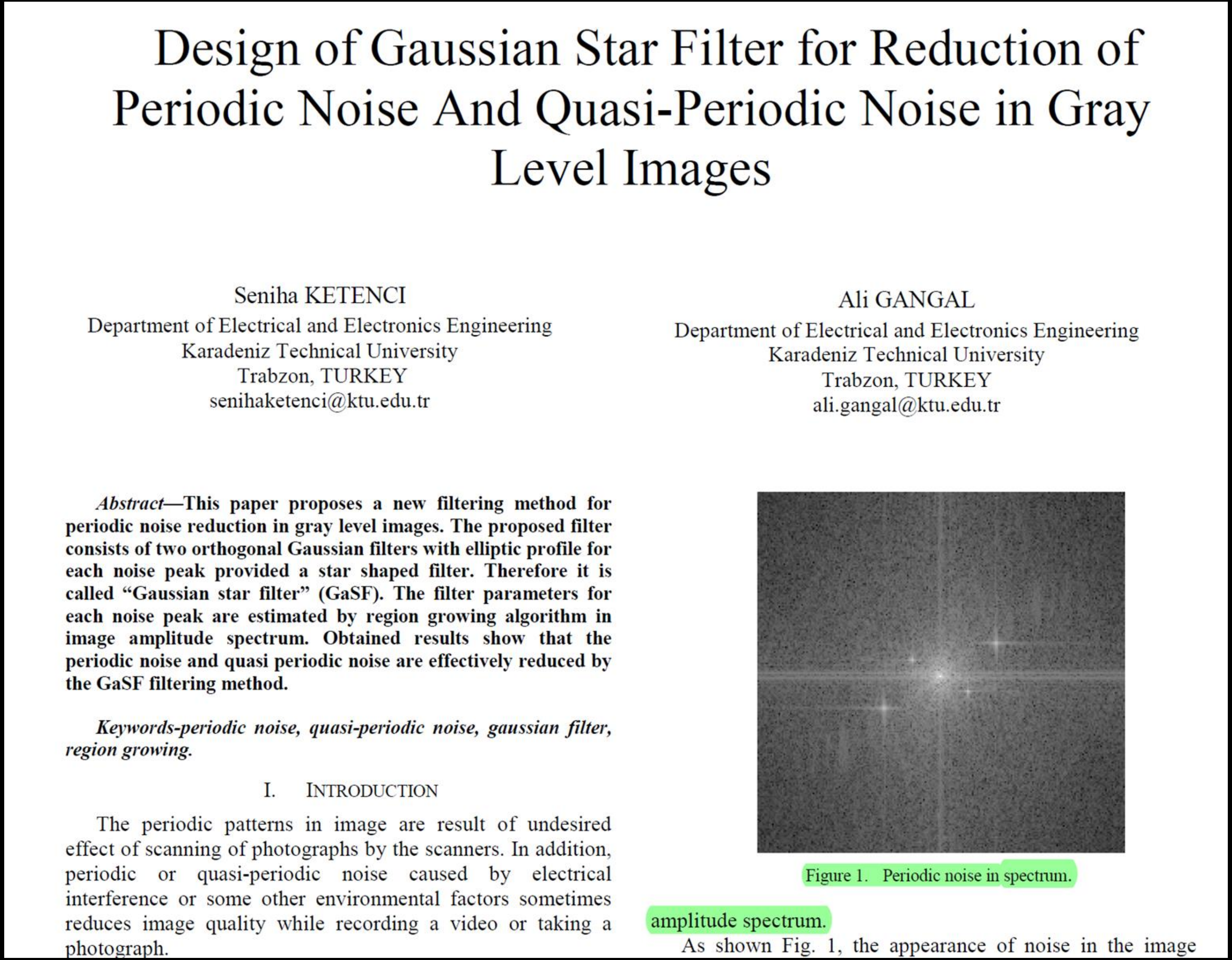
$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

$M \times N$ Image



- For FFT, $M = 2^m$, $N = 2^n$ $(m > 1, n > 1)$

Gaussian Star Low Pass Filter



Design of Gaussian Star Filter for Reduction of Periodic Noise And Quasi-Periodic Noise in Gray Level Images

Seniha KETENCI
Department of Electrical and Electronics Engineering
Karadeniz Technical University
Trabzon, TURKEY
senihaketenci@ktu.edu.tr

Ali GANGAL
Department of Electrical and Electronics Engineering
Karadeniz Technical University
Trabzon, TURKEY
ali.gangal@ktu.edu.tr

Abstract—**This paper proposes a new filtering method for periodic noise reduction in gray level images. The proposed filter consists of two orthogonal Gaussian filters with elliptic profile for each noise peak provided a star shaped filter. Therefore it is called "Gaussian star filter" (GaSF). The filter parameters for each noise peak are estimated by region growing algorithm in image amplitude spectrum. Obtained results show that the periodic noise and quasi periodic noise are effectively reduced by the GaSF filtering method.**

Keywords-periodic noise, quasi-periodic noise, gaussian filter, region growing.

I. INTRODUCTION

The periodic patterns in image are result of undesired effect of scanning of photographs by the scanners. In addition, periodic or quasi-periodic noise caused by electrical interference or some other environmental factors sometimes reduces image quality while recording a video or taking a photograph.

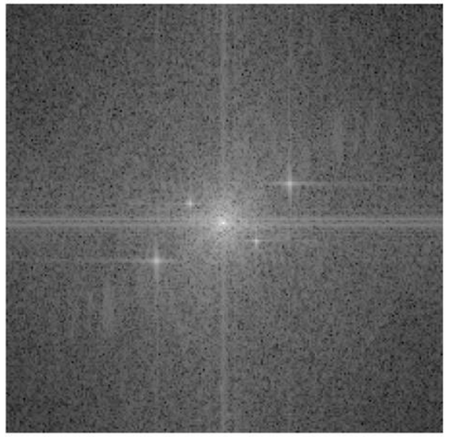


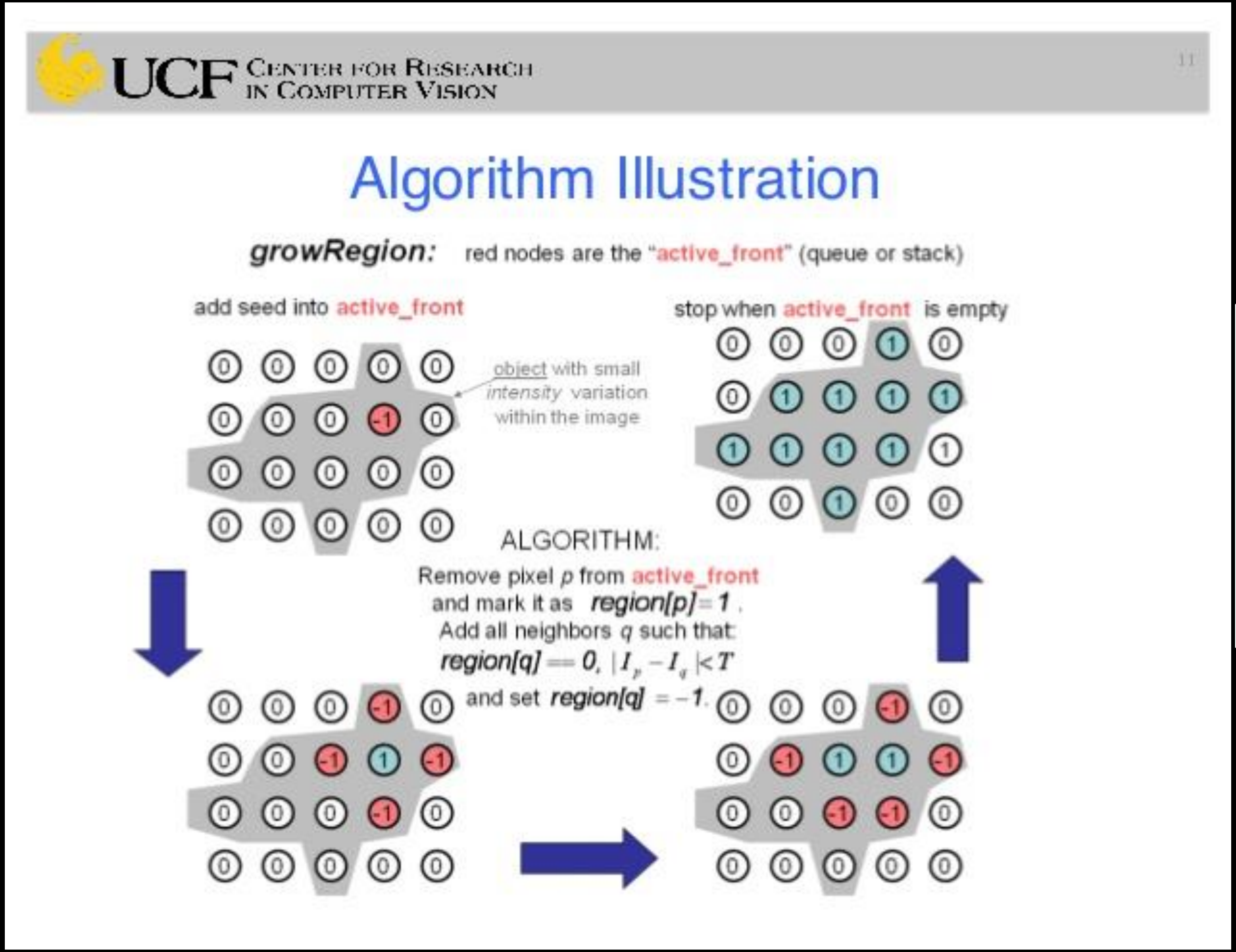
Figure 1. Periodic noise in spectrum.

amplitude spectrum.

As shown Fig. 1, the appearance of noise in the image

알고리즘

주파수 영역 분석 - Region Growing



UCF CENTER FOR RESEARCH IN COMPUTER VISION

11

Algorithm Illustration

growRegion: red nodes are the "active_front" (queue or stack)

add seed into active_front
object with small intensity variation within the image
stop when active_front is empty

ALGORITHM:

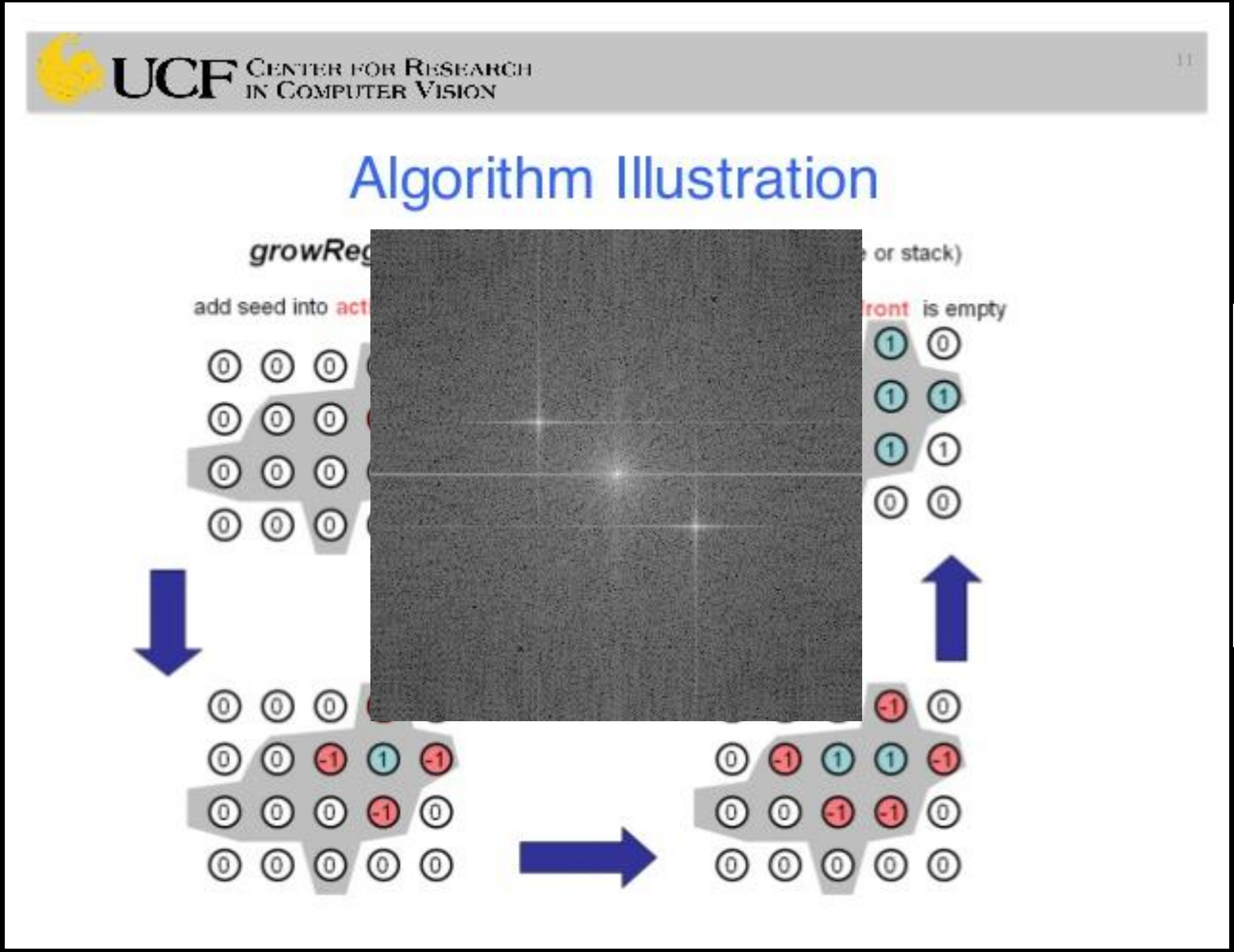
Remove pixel p from active_front and mark it as ***region[p]*** $= 1$.
Add all neighbors q such that:
region[q] $== 0, \ |I_p - I_q| < T$
and set ***region[q]*** $= -1$.

```
for every peak points
  while active points exist
    if valid
      mark it as blue zone (1)
      mark all inactive neighbors as active (-1)
    else
      mark it as inactive (0)
```

UCF Image Segmentation : Lecture 8

알고리즘

주파수 영역 분석 - Region Growing

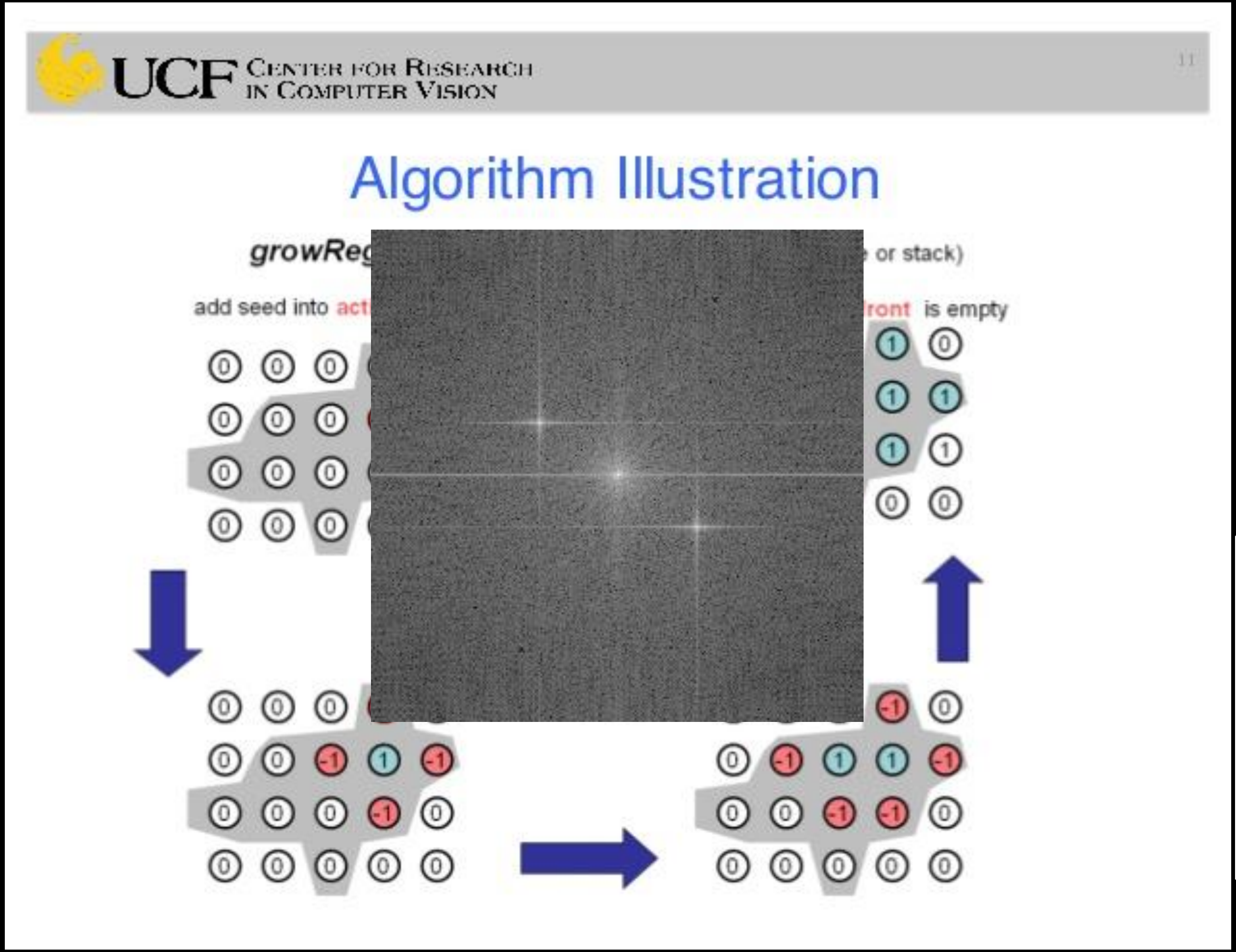


UCF Image Segmentation : Lecture 8

```
for every peak points
  while active points exist
    if valid
      mark it as blue zone (1)
      mark all inactive neighbors as active (-1)
    else
      mark it as inactive (0)
```

알고리즘

주파수 영역 분석 - Region Growing



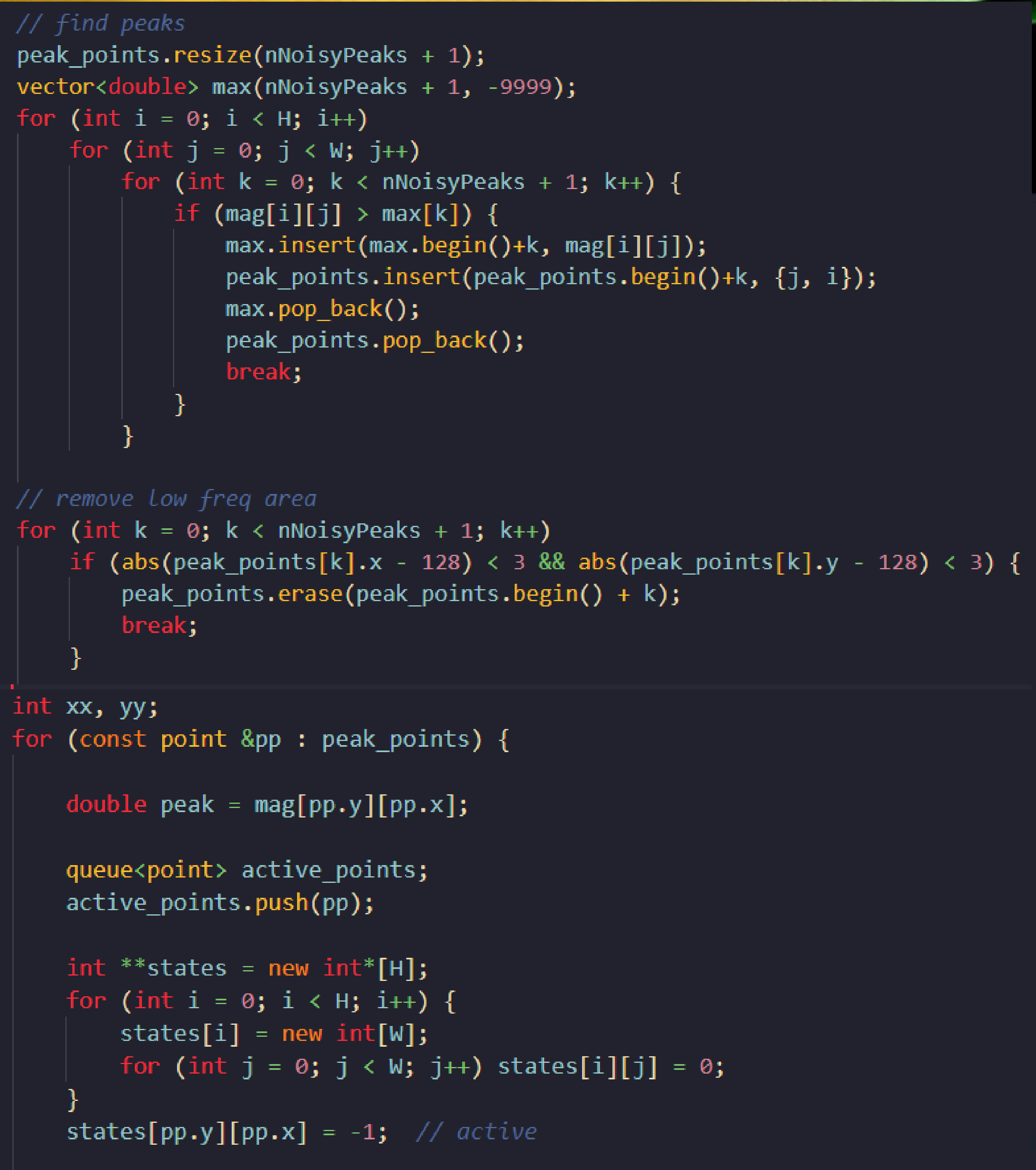
UCF Image Segmentation : Lecture 8

$$\begin{tikzcd}
\boxed{\text{peak points 찾기}} \arrow[d] \\
\boxed{\text{low freq peak 제거하기}} \arrow[d] \\
\text{pseudocode}
\end{tikzcd}$$

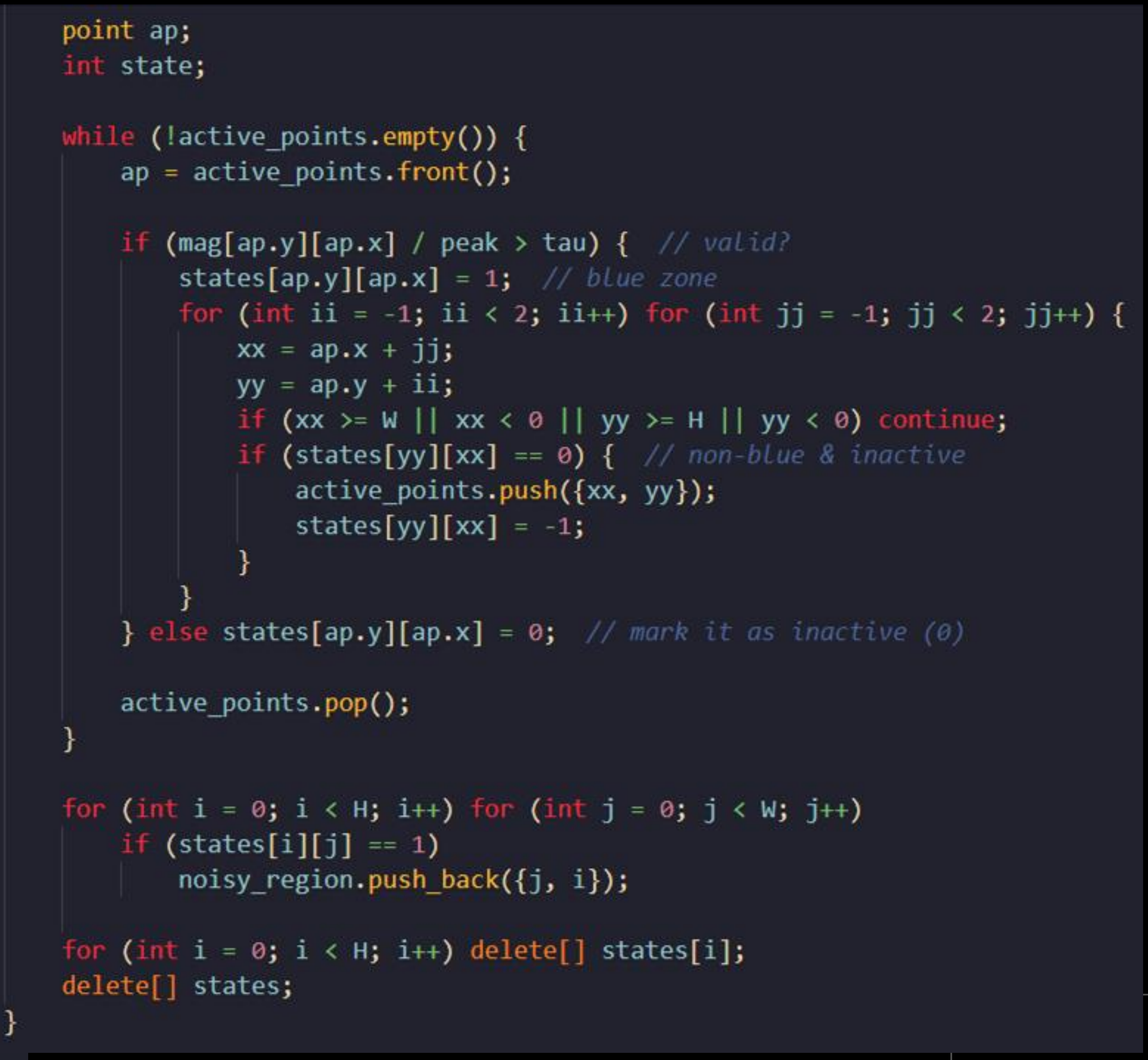
```
for every peak points
  while active points exist
    if valid
      mark it as blue zone (1)
      mark all inactive neighbors as active (-1)
    else
      mark it as inactive (0)
```

알고리즘

Region Growing

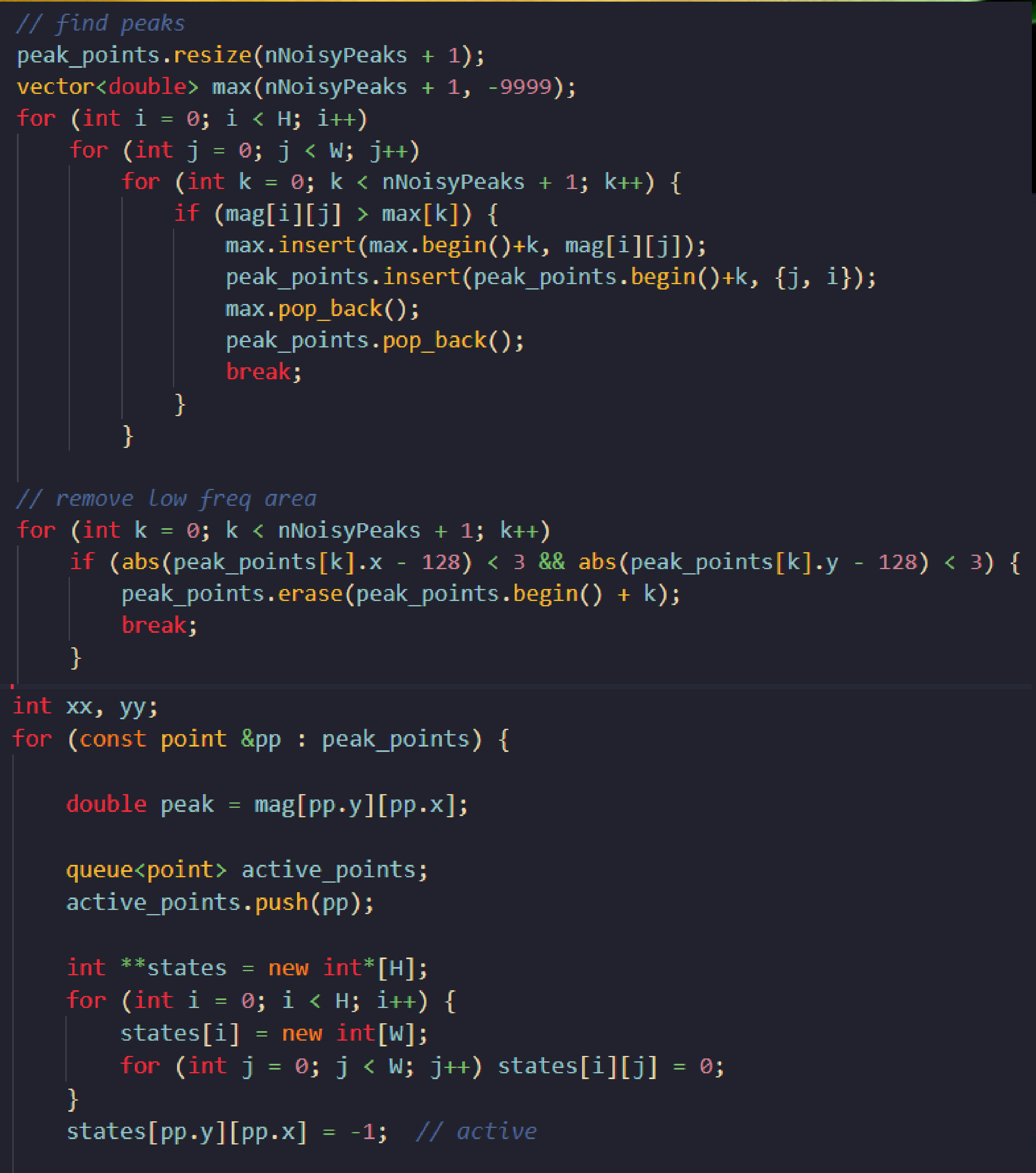


```cpp
// find peaks
peak_points.resize(nNoisyPeaks + 1);
vector<double> max(nNoisyPeaks + 1, -9999);
for (int i = 0; i < H; i++)
    for (int j = 0; j < W; j++)
        for (int k = 0; k < nNoisyPeaks + 1; k++) {
            if (mag[i][j] > max[k]) {
                max.insert(max.begin()+k, mag[i][j]);
                peak_points.insert(peak_points.begin()+k, {j, i});
                max.pop_back();
                peak_points.pop_back();
                break;
            }
        }

// remove low freq area
for (int k = 0; k < nNoisyPeaks + 1; k++)
    if (abs(peak_points[k].x - 128) < 3 && abs(peak_points[k].y - 128) < 3) {
        peak_points.erase(peak_points.begin() + k);
        break;
    }

int xx, yy;
for (const point &pp : peak_points) {

    double peak = mag[pp.y][pp.x];

    queue<point> active_points;
    active_points.push(pp);

    int **states = new int*[H];
    for (int i = 0; i < H; i++) {
        states[i] = new int[W];
        for (int j = 0; j < W; j++) states[i][j] = 0;
    }
    states[pp.y][pp.x] = -1;  // active
```

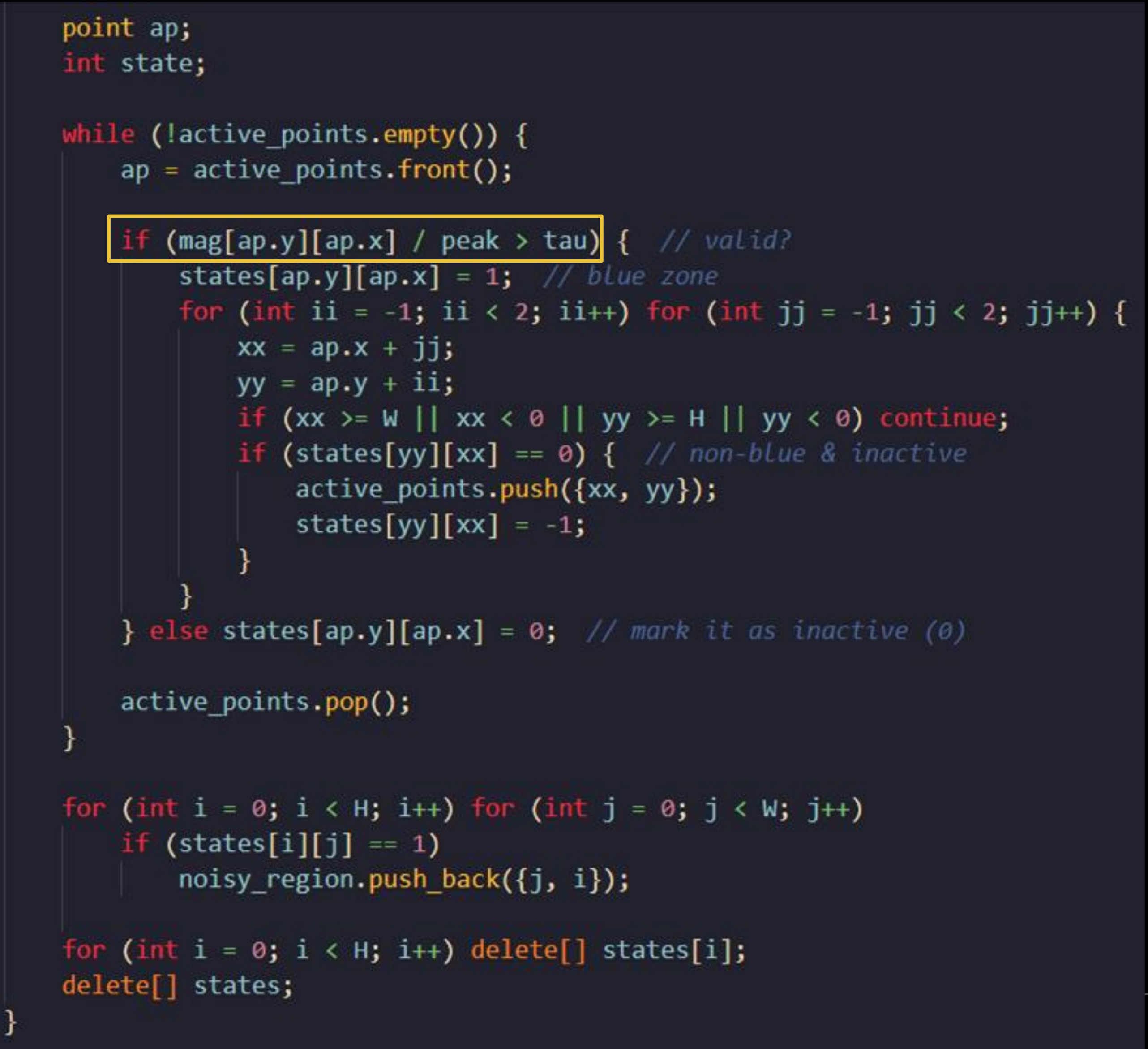


```cpp
    point ap;
    int state;

    while (!active_points.empty()) {
        ap = active_points.front();

        if (mag[ap.y][ap.x] / peak > tau) {  // valid?
            states[ap.y][ap.x] = 1;  // blue zone
            for (int ii = -1; ii < 2; ii++) for (int jj = -1; jj < 2; jj++) {
                xx = ap.x + jj;
                yy = ap.y + ii;
                if (xx >= W || xx < 0 || yy >= H || yy < 0) continue;
                if (states[yy][xx] == 0) {  // non-blue & inactive
                    active_points.push({xx, yy});
                    states[yy][xx] = -1;
                }
            }
        } else states[ap.y][ap.x] = 0;  // mark it as inactive (0)

        active_points.pop();
    }

    for (int i = 0; i < H; i++) for (int j = 0; j < W; j++)
        if (states[i][j] == 1)
            noisy_region.push_back({j, i});

    for (int i = 0; i < H; i++) delete[] states[i];
    delete[] states;
}
```

Region Growing

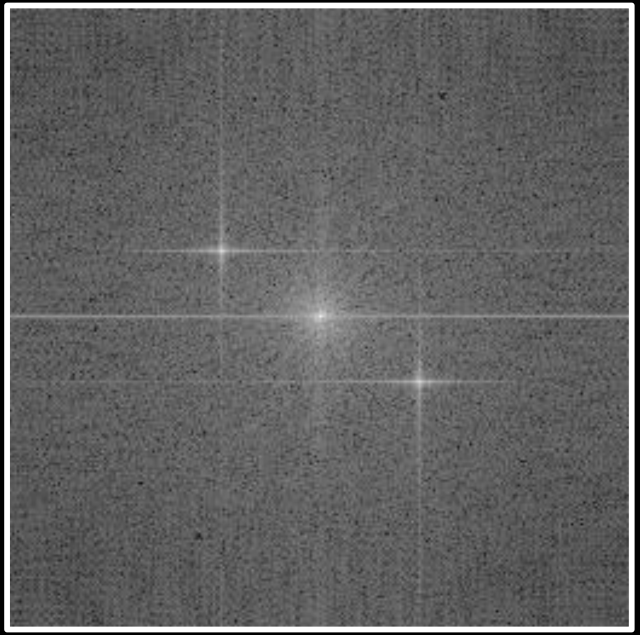


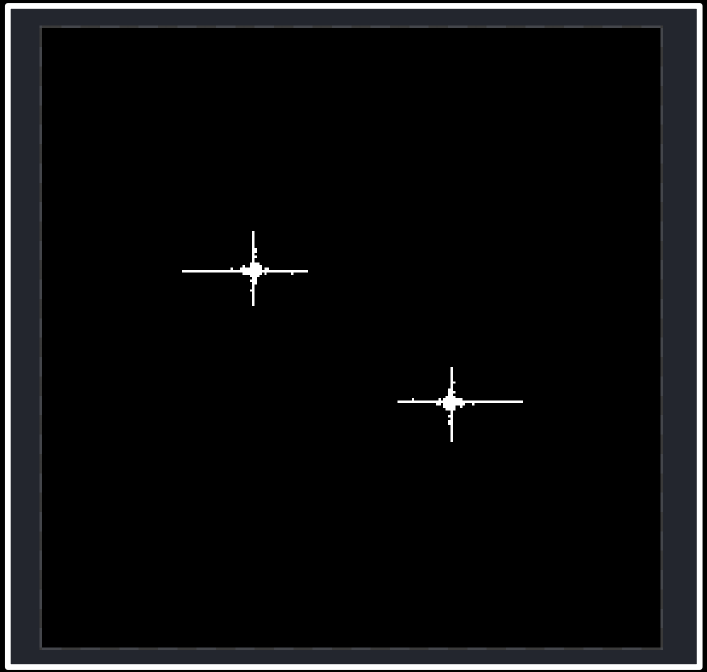
```cpp
// find peaks
peak_points.resize(nNoisyPeaks + 1);
vector<double> max(nNoisyPeaks + 1, -9999);
for (int i = 0; i < H; i++)
    for (int j = 0; j < W; j++)
        for (int k = 0; k < nNoisyPeaks + 1; k++) {
            if (mag[i][j] > max[k]) {
                max.insert(max.begin()+k, mag[i][j]);
                peak_points.insert(peak_points.begin()+k, {j, i});
                max.pop_back();
                peak_points.pop_back();
                break;
            }
        }

// remove low freq area
for (int k = 0; k < nNoisyPeaks + 1; k++)
    if (abs(peak_points[k].x - 128) < 3 && abs(peak_points[k].y - 128) < 3) {
        peak_points.erase(peak_points.begin() + k);
        break;
    }

int xx, yy;
for (const point &pp : peak_points) {

    double peak = mag[pp.y][pp.x];

    queue<point> active_points;
    active_points.push(pp);

    int **states = new int*[H];
    for (int i = 0; i < H; i++) {
        states[i] = new int[W];
        for (int j = 0; j < W; j++) states[i][j] = 0;
    }
    states[pp.y][pp.x] = -1;  // active
```



```cpp
    point ap;
    int state;

    while (!active_points.empty()) {
        ap = active_points.front();

        if (mag[ap.y][ap.x] / peak > tau) {  // valid?
            states[ap.y][ap.x] = 1;  // blue zone
            for (int ii = -1; ii < 2; ii++) for (int jj = -1; jj < 2; jj++) {
                xx = ap.x + jj;
                yy = ap.y + ii;
                if (xx >= W || xx < 0 || yy >= H || yy < 0) continue;
                if (states[yy][xx] == 0) {  // non-blue & inactive
                    active_points.push({xx, yy});
                    states[yy][xx] = -1;
                }
            }
        } else states[ap.y][ap.x] = 0;  // mark it as inactive (0)

        active_points.pop();
    }

    for (int i = 0; i < H; i++) for (int j = 0; j < W; j++)
        if (states[i][j] == 1)
            noisy_region.push_back({j, i});

    for (int i = 0; i < H; i++) delete[] states[i];
    delete[] states;
}
```

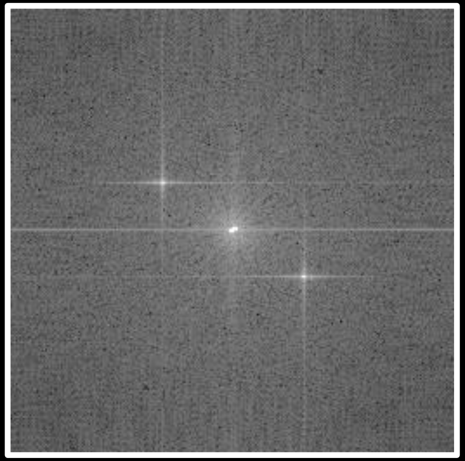
Region Growing

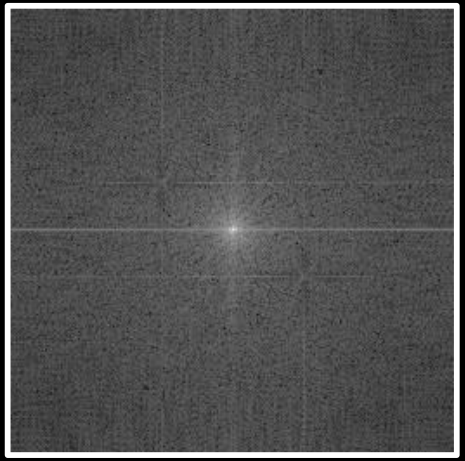




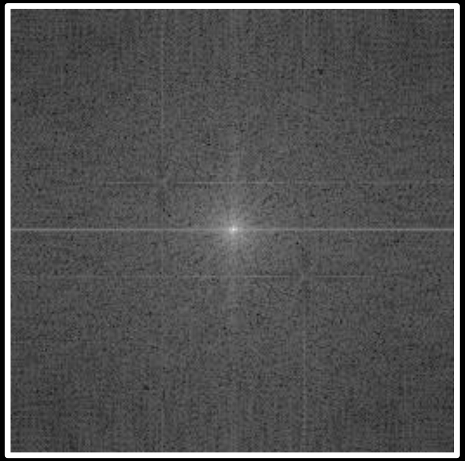
주파수 영역 필터링

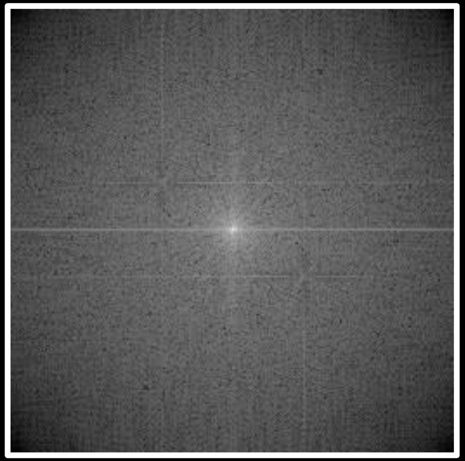
1. Kernel Function





2. 구형 Low Pass Filter





Kernel Function

$$GaSLF(u,v) = \begin{cases} \max(H_1, H_2) & \text{if } (H_1 \neq 0 \text{ and } H_2 \neq 0) \\ H_1 + H_2 & \text{else} \end{cases}$$

$$H_1(u,v) = \sum_{n=1}^{N} e^{-\frac{D_{1n}^2(u,v)}{2}}$$

$$H_2(u,v) = \sum_{n=1}^{N} e^{-\frac{D_{2n}^2(u,v)}{2}}$$

$$D_{1n}(u,v) = \left(\frac{u-u_n}{a_n}\right)^2 + \left(\frac{v-v_n}{b_n}\right)^2$$

$$D_{2n}(u,v) = \left(\frac{u-u_n}{b_n}\right)^2 + \left(\frac{v-v_n}{a_n}\right)^2$$

$$GaSF(u,v) = 1 - GaSLF(u,v)$$

Kernel Function

$$GaSLF(u,v) = \begin{cases} \max(H_1, H_2) & \text{if } (H_1 \neq 0 \text{ and } H_2 \neq 0) \\ H_1 + H_2 & \text{else} \end{cases}$$

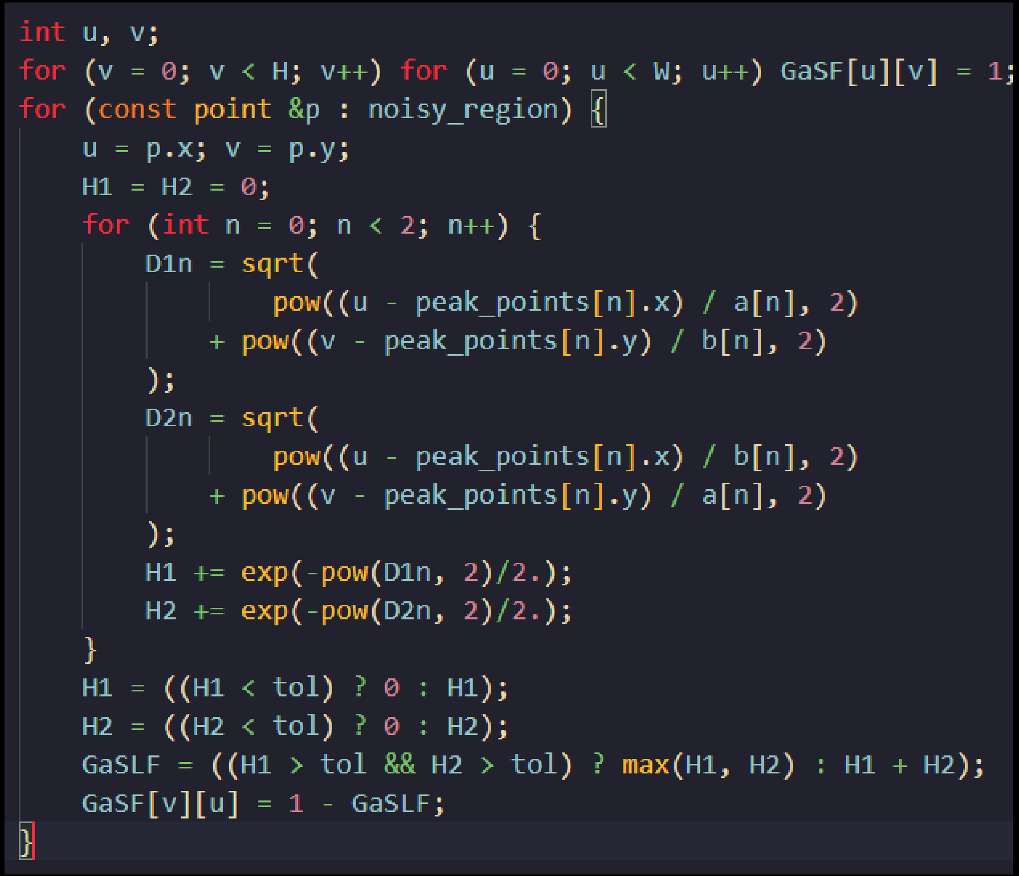
$$GaSF(u,v) = 1 - GaSLF(u,v)$$

$$H_1(u,v) = \sum_{n=1}^{N} e^{-\frac{D_{1n}^2(u,v)}{2}}$$

$$H_2(u,v) = \sum_{n=1}^{N} e^{-\frac{D_{2n}^2(u,v)}{2}}$$

$$D_{1n}(u,v) = \left(\frac{u-u_n}{a_n}\right)^2 + \left(\frac{v-v_n}{b_n}\right)^2$$

$$D_{2n}(u,v) = \left(\frac{u-u_n}{b_n}\right)^2 + \left(\frac{v-v_n}{a_n}\right)^2$$



```
int u, v;
for (v = 0; v < H; v++) for (u = 0; u < W; u++) GaSF[u][v] = 1;
for (const point &p : noisy_region) {
    u = p.x; v = p.y;
    H1 = H2 = 0;
    for (int n = 0; n < 2; n++) {
        D1n = sqrt(
                pow((u - peak_points[n].x) / a[n], 2)
              + pow((v - peak_points[n].y) / b[n], 2)
        );
        D2n = sqrt(
                pow((u - peak_points[n].x) / b[n], 2)
              + pow((v - peak_points[n].y) / a[n], 2)
        );
        H1 += exp(-pow(D1n, 2)/2.);
        H2 += exp(-pow(D2n, 2)/2.);
    }
    H1 = ((H1 < tol) ? 0 : H1);
    H2 = ((H2 < tol) ? 0 : H2);
    GaSLF = ((H1 > tol && H2 > tol) ? max(H1, H2) : H1 + H2);
    GaSF[v][u] = 1 - GaSLF;
}
```

알고리즘

Kernel Function

$$GaSLF(u,v) = \begin{cases} \max(H_1, H_2) & \text{if } (H_1 \neq 0 \text{ and } H_2 \neq 0) \\ H_1 + H_2 & \text{else} \end{cases}$$

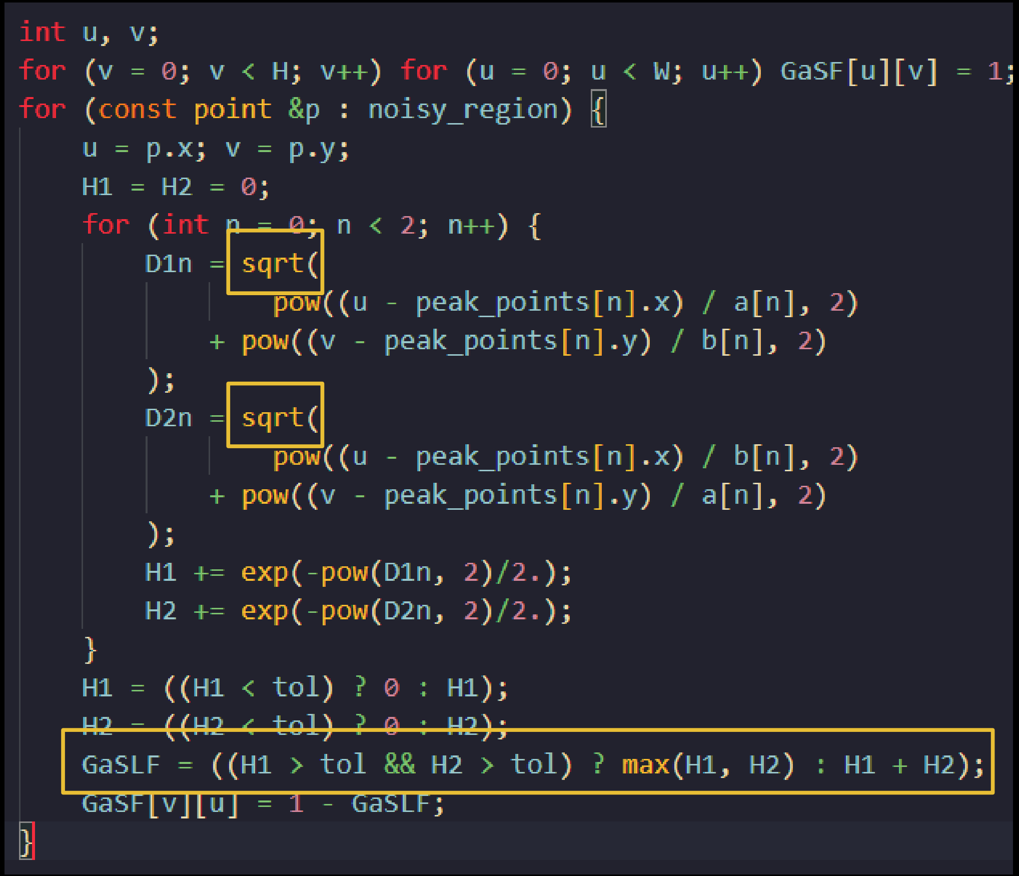
$$GaSF(u,v) = 1 - GaSLF(u,v)$$

$$H_1(u,v) = \sum_{n=1}^{N} e^{-\frac{D_{1n}^2(u,v)}{2}}$$

$$H_2(u,v) = \sum_{n=1}^{N} e^{-\frac{D_{2n}^2(u,v)}{2}}$$

$$D_{1n}(u,v) = \left(\frac{u-u_n}{a_n}\right)^2 + \left(\frac{v-v_n}{b_n}\right)^2$$

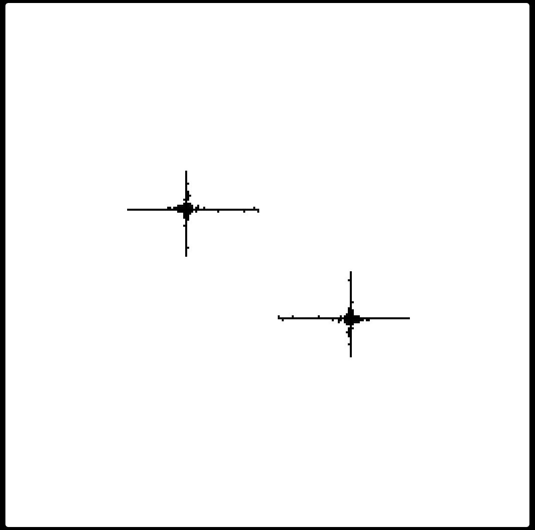
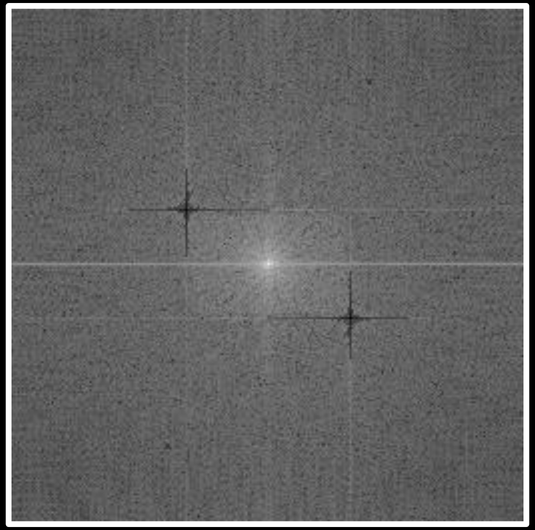
$$D_{2n}(u,v) = \left(\frac{u-u_n}{b_n}\right)^2 + \left(\frac{v-v_n}{a_n}\right)^2$$



```
int u, v;
for (v = 0; v < H; v++) for (u = 0; u < W; u++) GaSF[u][v] = 1;
for (const point &p : noisy_region) {
    u = p.x; v = p.y;
    H1 = H2 = 0;
    for (int n = 0; n < 2; n++) {
        D1n = sqrt(
                pow((u - peak_points[n].x) / a[n], 2)
              + pow((v - peak_points[n].y) / b[n], 2)
        );
        D2n = sqrt(
                pow((u - peak_points[n].x) / b[n], 2)
              + pow((v - peak_points[n].y) / a[n], 2)
        );
        H1 += exp(-pow(D1n, 2)/2.);
        H2 += exp(-pow(D2n, 2)/2.);
    }
    H1 = ((H1 < tol) ? 0 : H1);
    H2 = ((H2 < tol) ? 0 : H2);
    GaSLF = ((H1 > tol && H2 > tol) ? max(H1, H2) : H1 + H2);
    GaSF[v][u] = 1 - GaSLF;
}
```

알고리즘

Kernel Function

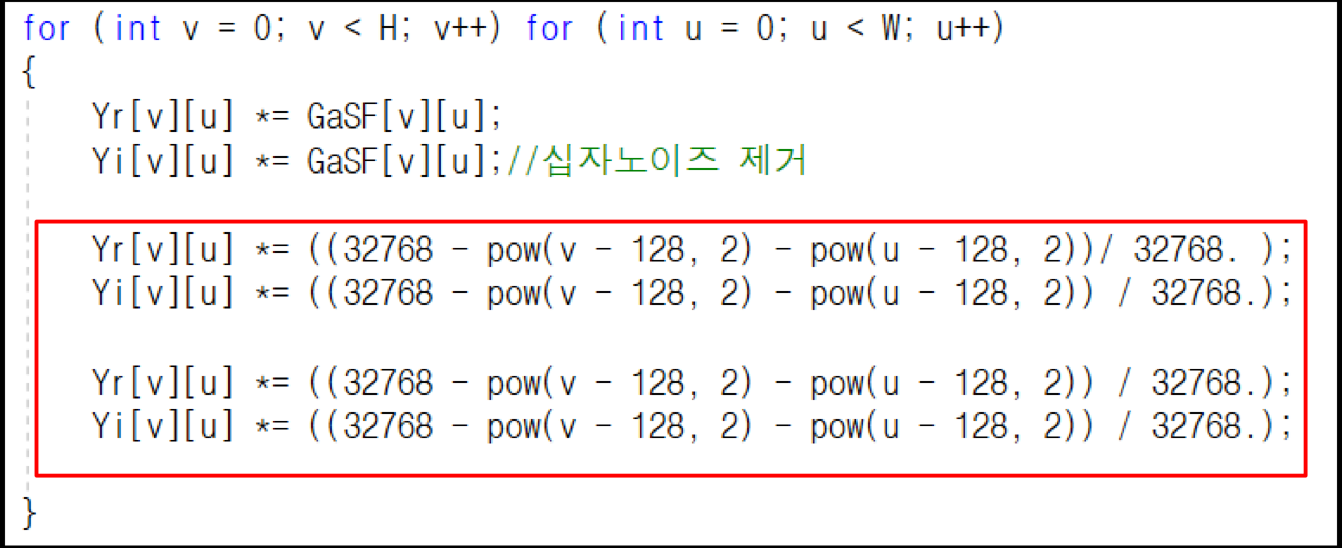
구형 LPF

$$\tilde{F}(u,v) = \left(1 - \frac{(u-128)^2 + (v-128)^2}{2 \times 128^2}\right) F(u,v)$$

이미지의 중심(128, 128)에서 최대값

중심->바깥 : mag감소 (LPF)

성능 향상을 위해 2번 실행

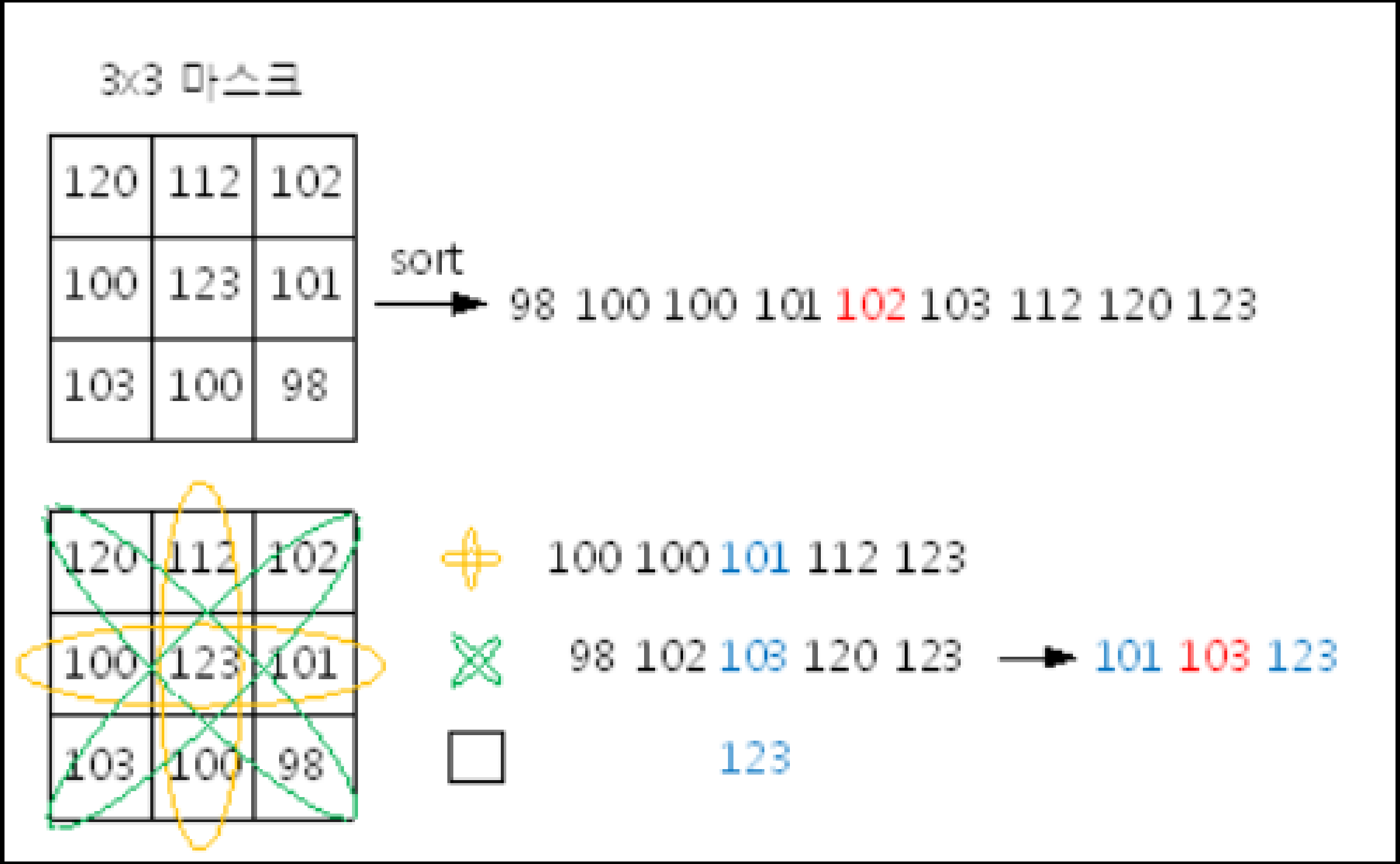


```
for (int v = 0; v < H; v++) for (int u = 0; u < W; u++)
{
    Yr[v][u] *= GaSF[v][u];
    Yi[v][u] *= GaSF[v][u];//십자노이즈 제거

    Yr[v][u] *= ((32768 - pow(v - 128, 2) - pow(u - 128, 2))/ 32768. );
    Yi[v][u] *= ((32768 - pow(v - 128, 2) - pow(u - 128, 2)) / 32768.);

    Yr[v][u] *= ((32768 - pow(v - 128, 2) - pow(u - 128, 2)) / 32768.);
    Yi[v][u] *= ((32768 - pow(v - 128, 2) - pow(u - 128, 2)) / 32768.);

}
```

IFFT

$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

Hybrid Median Filter



1. 마스크 내의 <u>전체 픽셀에 대한 중간값</u>

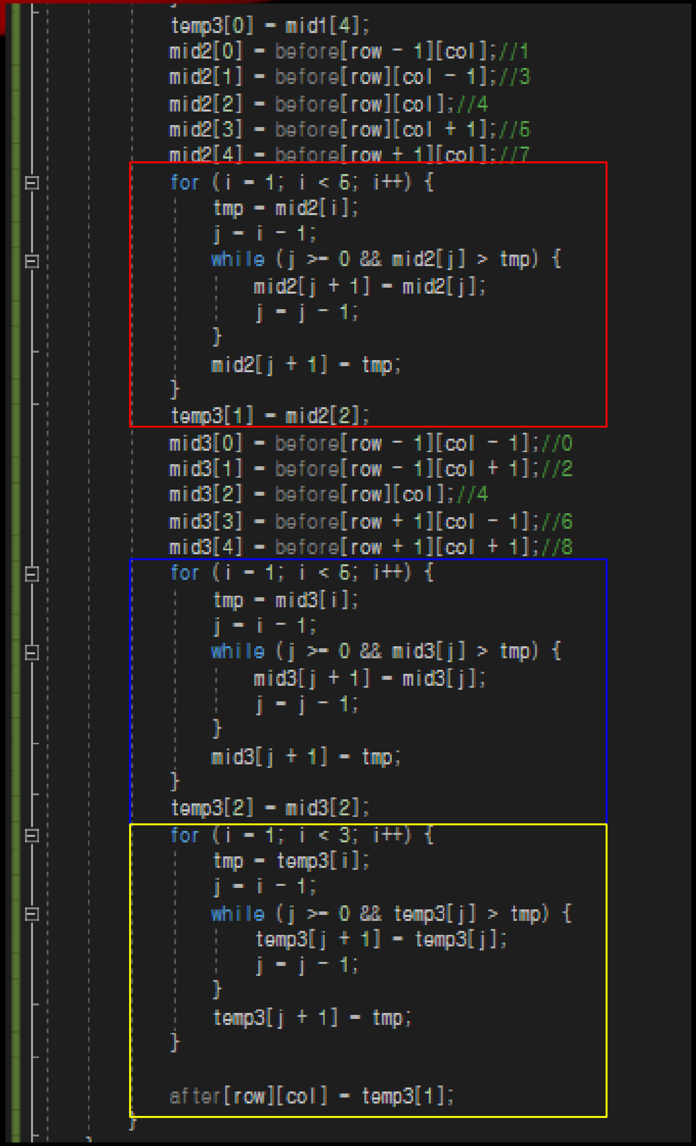
2. [1,3,4,5,7]에 위치한 픽셀에 대한 중간값

3. [0,2,4,6,8]에 위치한 픽셀에 대한 중간값

4. 구한 3개의 값들의 중간값을 [4]에 입력

Trial

Hybrid Median Filter



```
        temp3[0] = mid1[4];
        mid2[0] = before[row - 1][col];//1
        mid2[1] = before[row][col - 1];//3
        mid2[2] = before[row][col];//4
        mid2[3] = before[row][col + 1];//5
        mid2[4] = before[row + 1][col];//7
        for (i = 1; i < 5; i++) {
            tmp = mid2[i];
            j = i - 1;
            while (j >= 0 && mid2[j] > tmp) {
                mid2[j + 1] = mid2[j];
                j = j - 1;
            }
            mid2[j + 1] = tmp;
        }
        temp3[1] = mid2[2];
        mid3[0] = before[row - 1][col - 1];//0
        mid3[1] = before[row - 1][col + 1];//2
        mid3[2] = before[row][col];//4
        mid3[3] = before[row + 1][col - 1];//6
        mid3[4] = before[row + 1][col + 1];//8
        for (i = 1; i < 5; i++) {
            tmp = mid3[i];
            j = i - 1;
            while (j >= 0 && mid3[j] > tmp) {
                mid3[j + 1] = mid3[j];
                j = j - 1;
            }
            mid3[j + 1] = tmp;
        }
        temp3[2] = mid3[2];
        for (i = 1; i < 3; i++) {
            tmp = temp3[i];
            j = i - 1;
            while (j >= 0 && temp3[j] > tmp) {
                temp3[j + 1] = temp3[j];
                j = j - 1;
            }
            temp3[j + 1] = tmp;
        }

        after[row][col] = temp3[1];
    }
```

3종류의 알고리즘으로

얻은 3개의 값들 중의

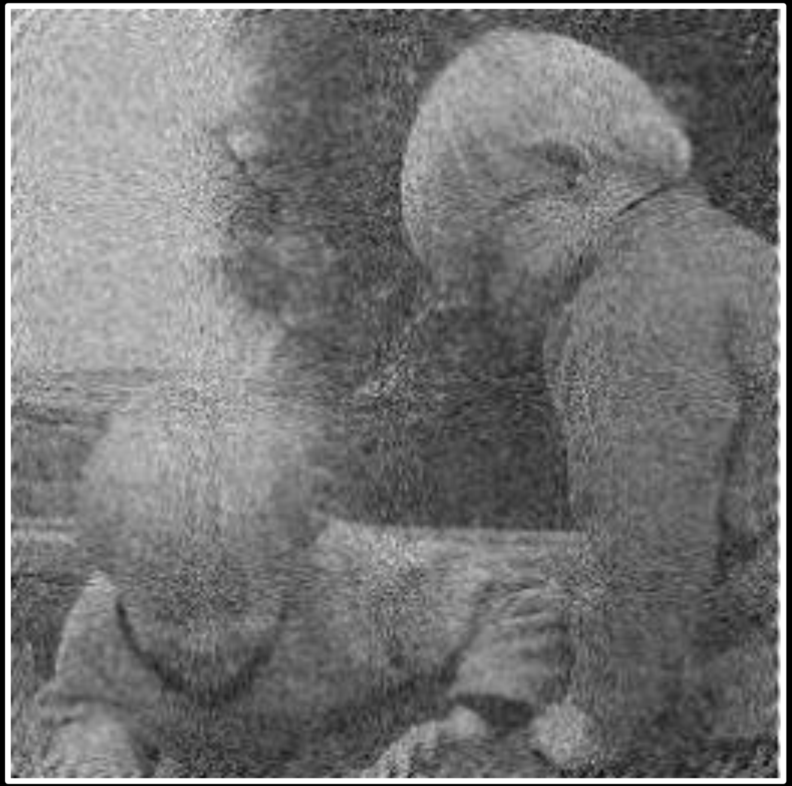
중간값을 고르는 함수 구현

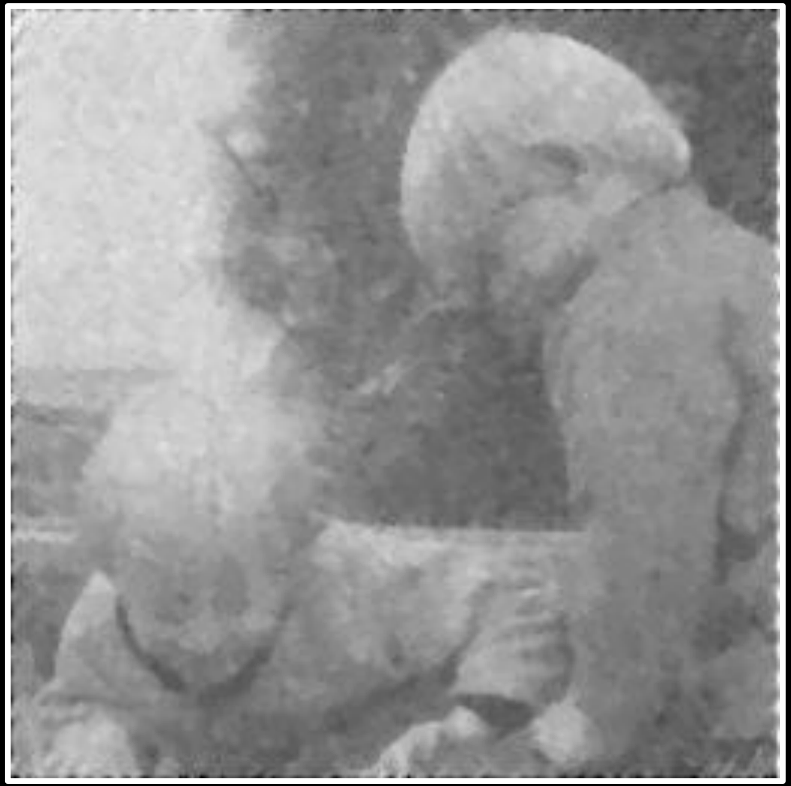
기댓값)

salt & pepper noise와 유사한

노이즈 제거

Hybrid Median Filter

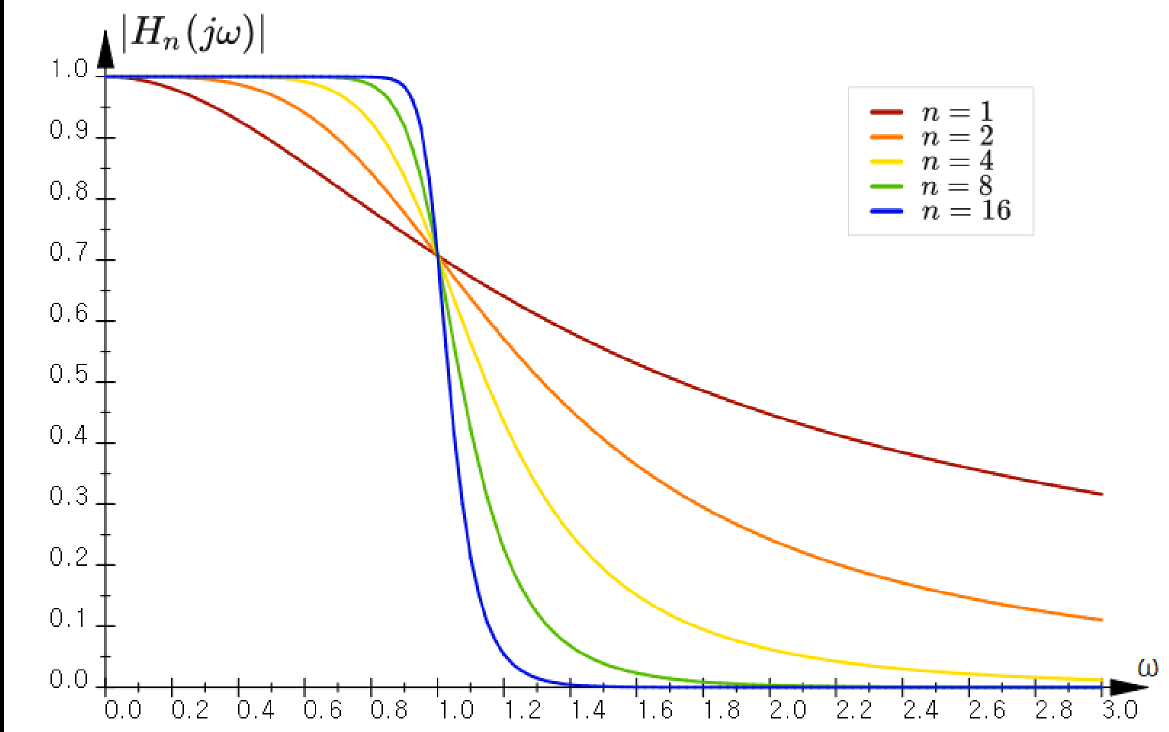




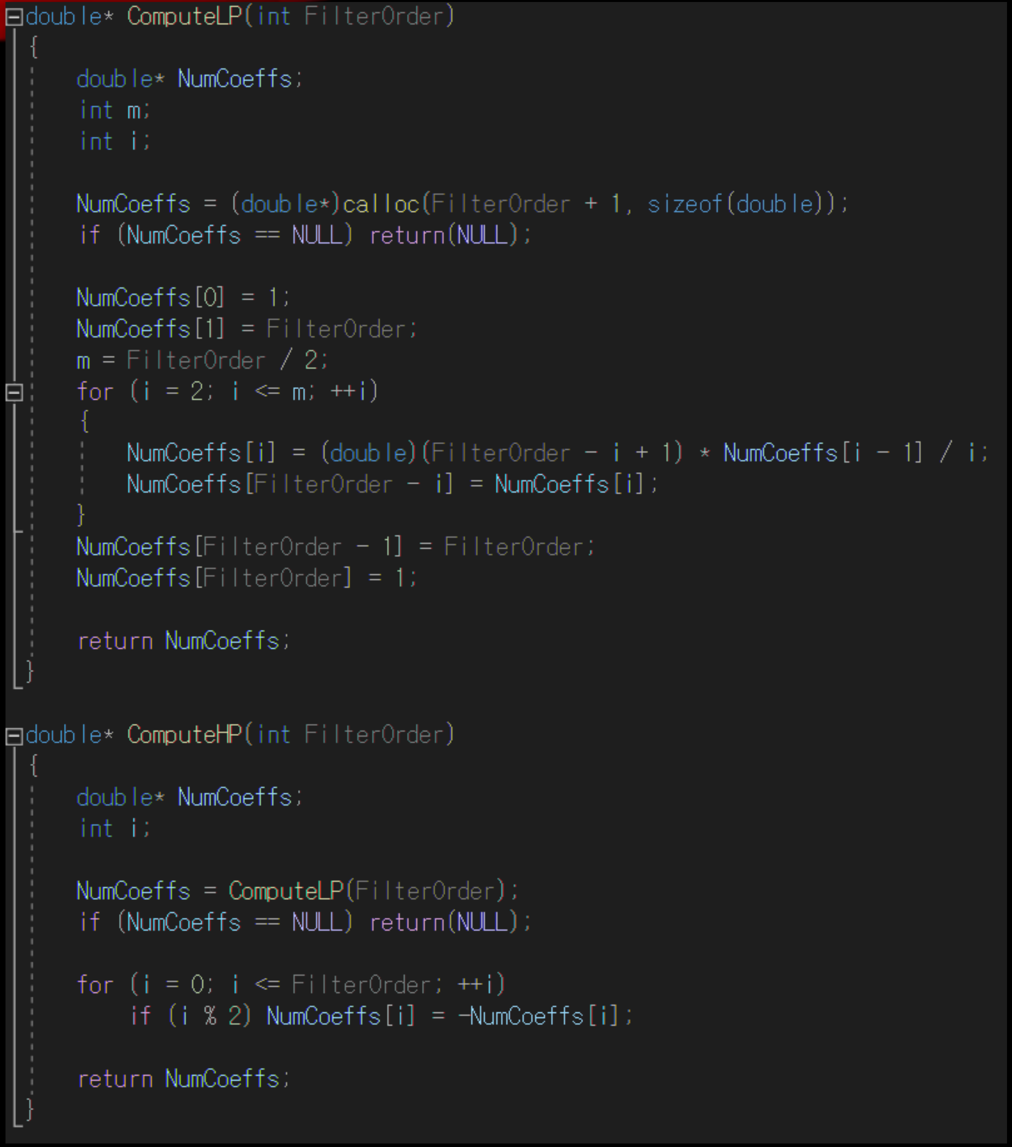
Noise 뿐만 아니라 edge 또한 손상

Butter worth filter

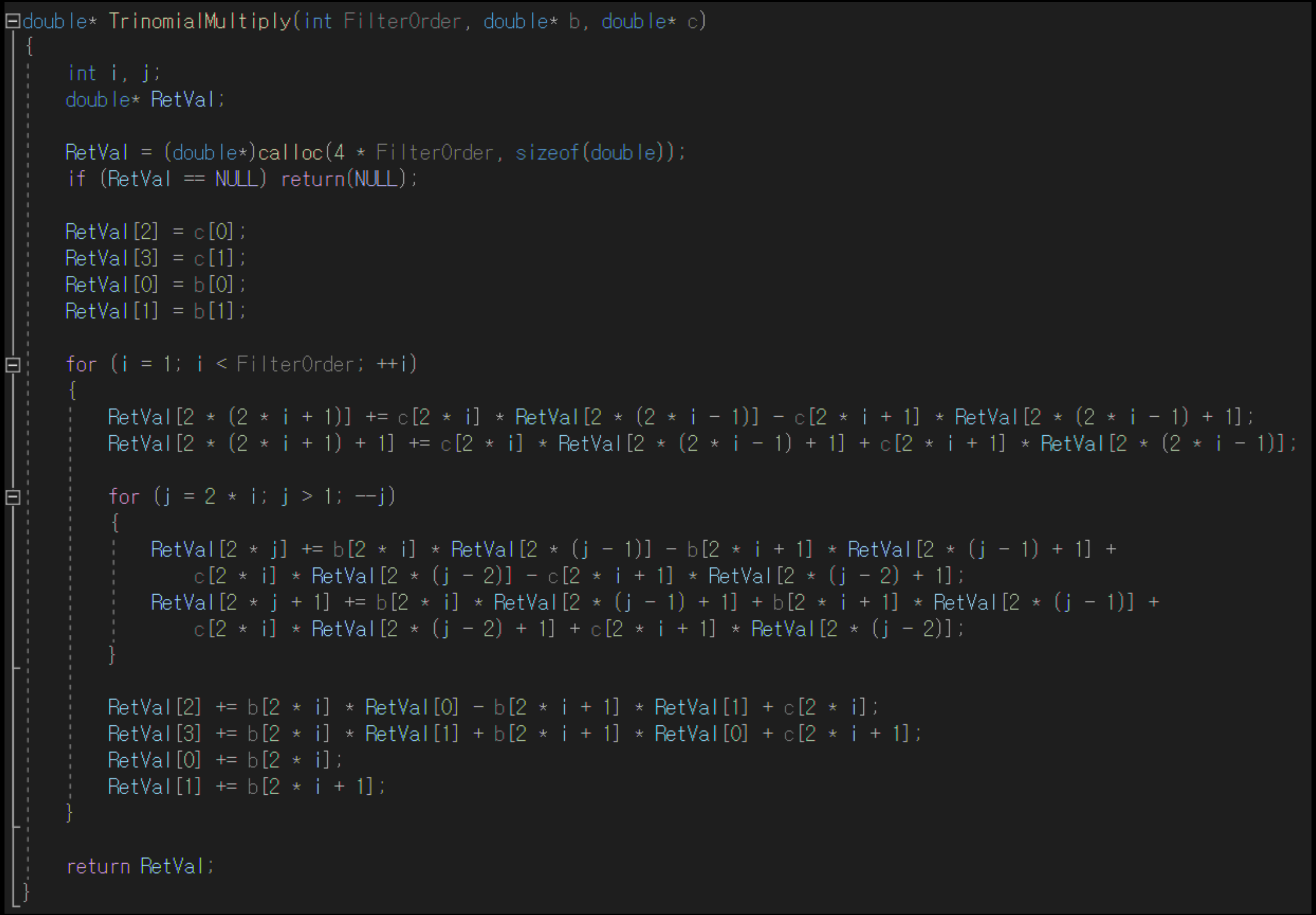
$$|H_n(j\omega)| \triangleq \frac{1}{\sqrt{1+\omega^{2n}}}$$



Butter worth filter



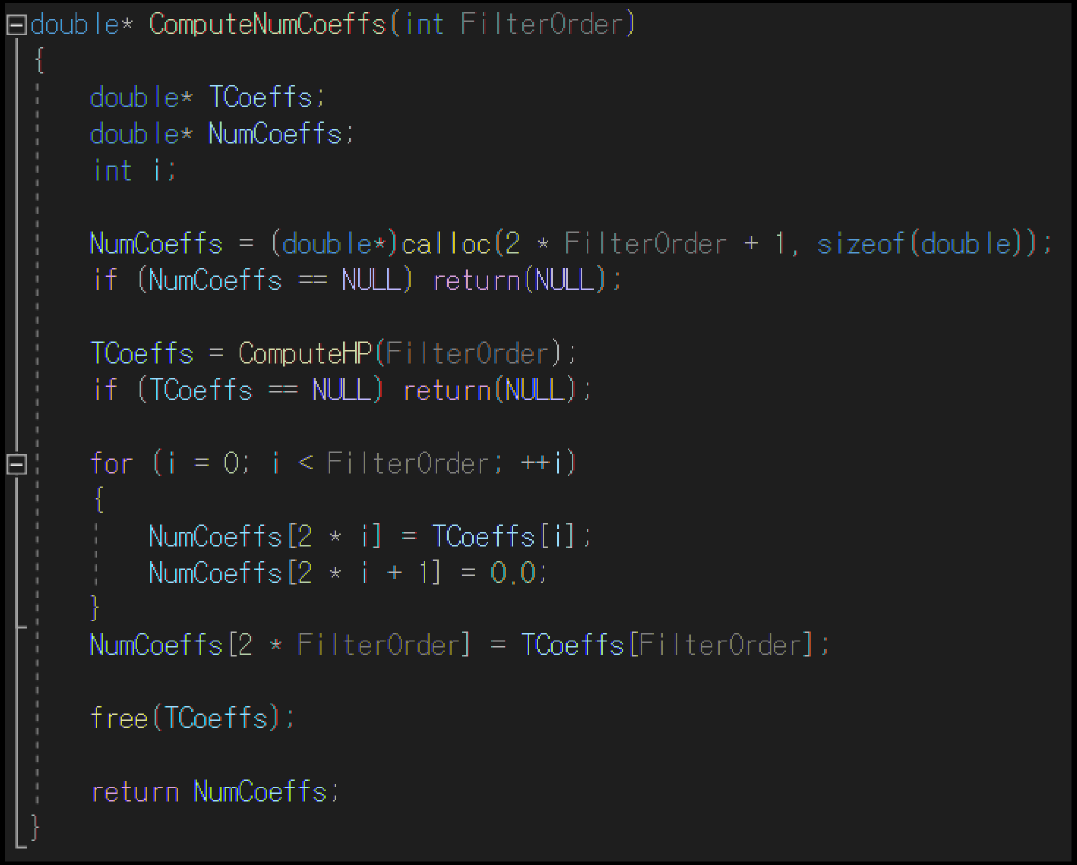
```
double* ComputeLP(int FilterOrder)
{
    double* NumCoeffs;
    int m;
    int i;

    NumCoeffs = (double*)calloc(FilterOrder + 1, sizeof(double));
    if (NumCoeffs == NULL) return(NULL);

    NumCoeffs[0] = 1;
    NumCoeffs[1] = FilterOrder;
    m = FilterOrder / 2;
    for (i = 2; i <= m; ++i)
    {
        NumCoeffs[i] = (double)(FilterOrder - i + 1) * NumCoeffs[i - 1] / i;
        NumCoeffs[FilterOrder - i] = NumCoeffs[i];
    }
    NumCoeffs[FilterOrder - 1] = FilterOrder;
    NumCoeffs[FilterOrder] = 1;

    return NumCoeffs;
}

double* ComputeHP(int FilterOrder)
{
    double* NumCoeffs;
    int i;

    NumCoeffs = ComputeLP(FilterOrder);
    if (NumCoeffs == NULL) return(NULL);

    for (i = 0; i <= FilterOrder; ++i)
        if (i % 2) NumCoeffs[i] = -NumCoeffs[i];

    return NumCoeffs;
}
```



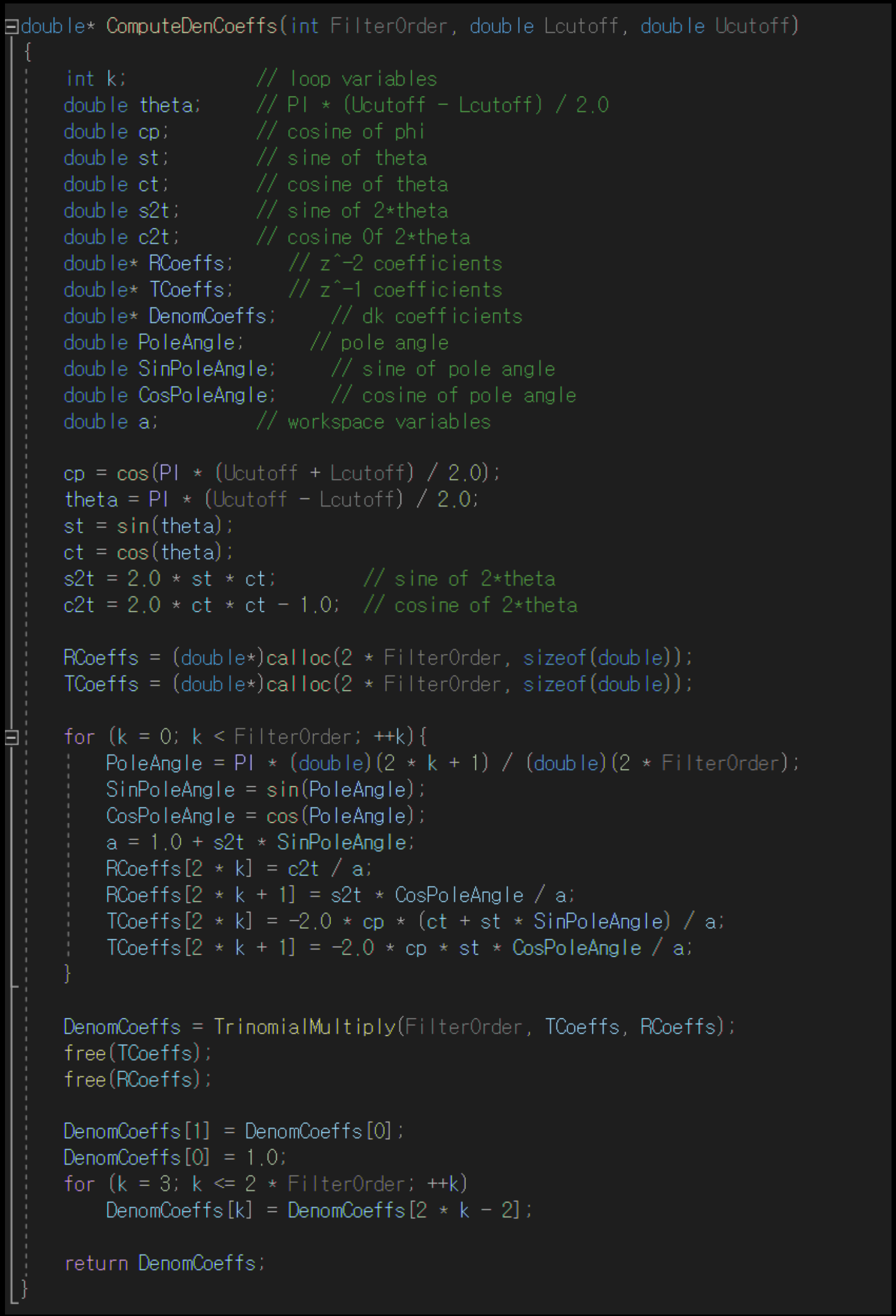
```
double* TrinomialMultiply(int FilterOrder, double* b, double* c)
{
    int i, j;
    double* RetVal;

    RetVal = (double*)calloc(4 * FilterOrder, sizeof(double));
    if (RetVal == NULL) return(NULL);

    RetVal[2] = c[0];
    RetVal[3] = c[1];
    RetVal[0] = b[0];
    RetVal[1] = b[1];

    for (i = 1; i < FilterOrder; ++i)
    {
        RetVal[2 * (2 * i + 1)] += c[2 * i] * RetVal[2 * (2 * i - 1)] - c[2 * i + 1] * RetVal[2 * (2 * i - 1) + 1];
        RetVal[2 * (2 * i + 1) + 1] += c[2 * i] * RetVal[2 * (2 * i - 1) + 1] + c[2 * i + 1] * RetVal[2 * (2 * i - 1)];

        for (j = 2 * i; j > 1; --j)
        {
            RetVal[2 * j] += b[2 * i] * RetVal[2 * (j - 1)] - b[2 * i + 1] * RetVal[2 * (j - 1) + 1] +
                c[2 * i] * RetVal[2 * (j - 2)] - c[2 * i + 1] * RetVal[2 * (j - 2) + 1];
            RetVal[2 * j + 1] += b[2 * i] * RetVal[2 * (j - 1) + 1] + b[2 * i + 1] * RetVal[2 * (j - 1)] +
                c[2 * i] * RetVal[2 * (j - 2) + 1] + c[2 * i + 1] * RetVal[2 * (j - 2)];
        }

        RetVal[2] += b[2 * i] * RetVal[0] - b[2 * i + 1] * RetVal[1] + c[2 * i];
        RetVal[3] += b[2 * i] * RetVal[1] + b[2 * i + 1] * RetVal[0] + c[2 * i + 1];
        RetVal[0] += b[2 * i];
        RetVal[1] += b[2 * i + 1];
    }

    return RetVal;
}
```

알고리즘

Butter worth filter

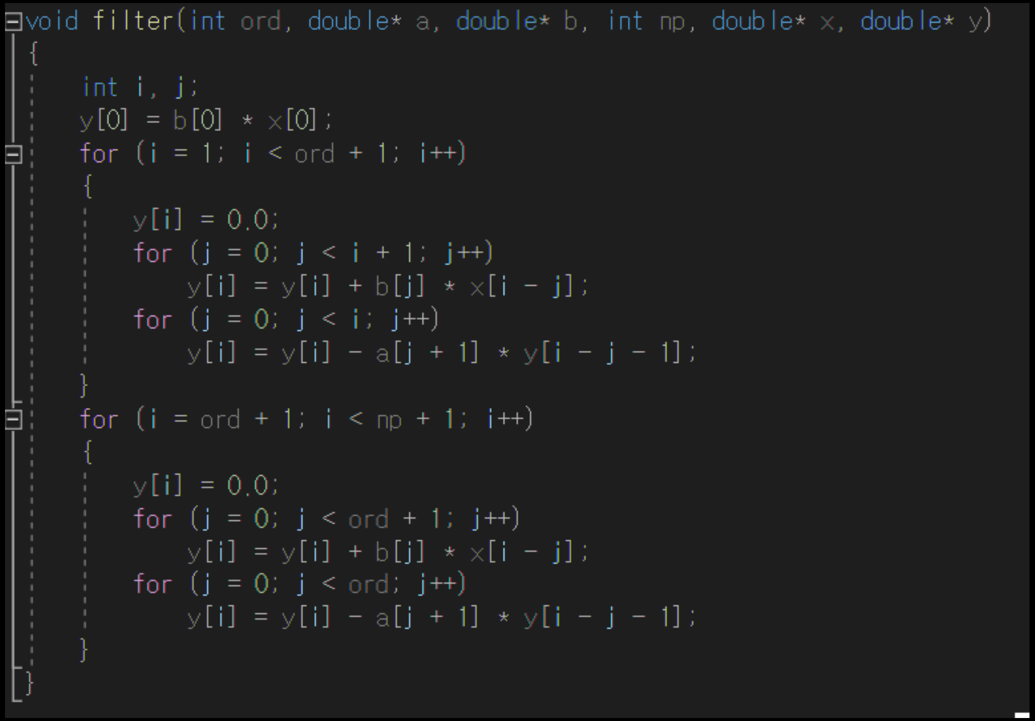


```
double* ComputeNumCoeffs(int FilterOrder)
{
    double* TCoeffs;
    double* NumCoeffs;
    int i;

    NumCoeffs = (double*)calloc(2 * FilterOrder + 1, sizeof(double));
    if (NumCoeffs == NULL) return(NULL);

    TCoeffs = ComputeHP(FilterOrder);
    if (TCoeffs == NULL) return(NULL);

    for (i = 0; i < FilterOrder; ++i)
    {
        NumCoeffs[2 * i] = TCoeffs[i];
        NumCoeffs[2 * i + 1] = 0.0;
    }
    NumCoeffs[2 * FilterOrder] = TCoeffs[FilterOrder];

    free(TCoeffs);

    return NumCoeffs;
}
```

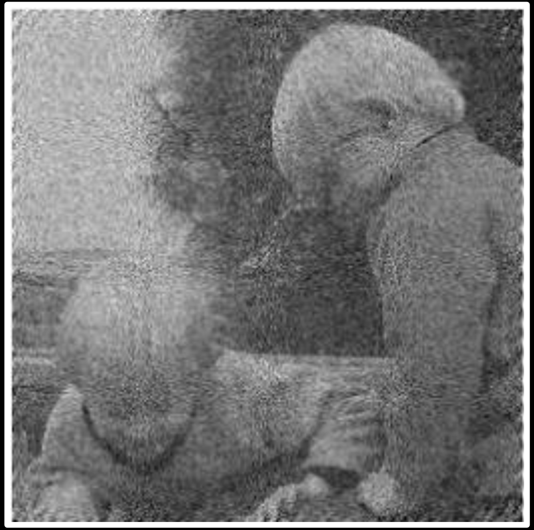


```
double* ComputeDenCoeffs(int FilterOrder, double Lcutoff, double Ucutoff)
{
    int k;              // loop variables
    double theta;       // PI * (Ucutoff - Lcutoff) / 2.0
    double cp;          // cosine of phi
    double st;          // sine of theta
    double ct;          // cosine of theta
    double s2t;         // sine of 2*theta
    double c2t;         // cosine Of 2*theta
    double* RCoeffs;      // z^-2 coefficients
    double* TCoeffs;      // z^-1 coefficients
    double* DenomCoeffs;      // dk coefficients
    double PoleAngle;       // pole angle
    double SinPoleAngle;      // sine of pole angle
    double CosPoleAngle;      // cosine of pole angle
    double a;           // workspace variables

    cp = cos(PI * (Ucutoff + Lcutoff) / 2.0);
    theta = PI * (Ucutoff - Lcutoff) / 2.0;
    st = sin(theta);
    ct = cos(theta);
    s2t = 2.0 * st * ct;        // sine of 2*theta
    c2t = 2.0 * ct * ct - 1.0;  // cosine of 2*theta

    RCoeffs = (double*)calloc(2 * FilterOrder, sizeof(double));
    TCoeffs = (double*)calloc(2 * FilterOrder, sizeof(double));

    for (k = 0; k < FilterOrder; ++k){
        PoleAngle = PI * (double)(2 * k + 1) / (double)(2 * FilterOrder);
        SinPoleAngle = sin(PoleAngle);
        CosPoleAngle = cos(PoleAngle);
        a = 1.0 + s2t * SinPoleAngle;
        RCoeffs[2 * k] = c2t / a;
        RCoeffs[2 * k + 1] = s2t * CosPoleAngle / a;
        TCoeffs[2 * k] = -2.0 * cp * (ct + st * SinPoleAngle) / a;
        TCoeffs[2 * k + 1] = -2.0 * cp * st * CosPoleAngle / a;
    }

    DenomCoeffs = TrinomialMultiply(FilterOrder, TCoeffs, RCoeffs);
    free(TCoeffs);
    free(RCoeffs);

    DenomCoeffs[1] = DenomCoeffs[0];
    DenomCoeffs[0] = 1.0;
    for (k = 3; k <= 2 * FilterOrder; ++k)
        DenomCoeffs[k] = DenomCoeffs[2 * k - 2];

    return DenomCoeffs;
}
```

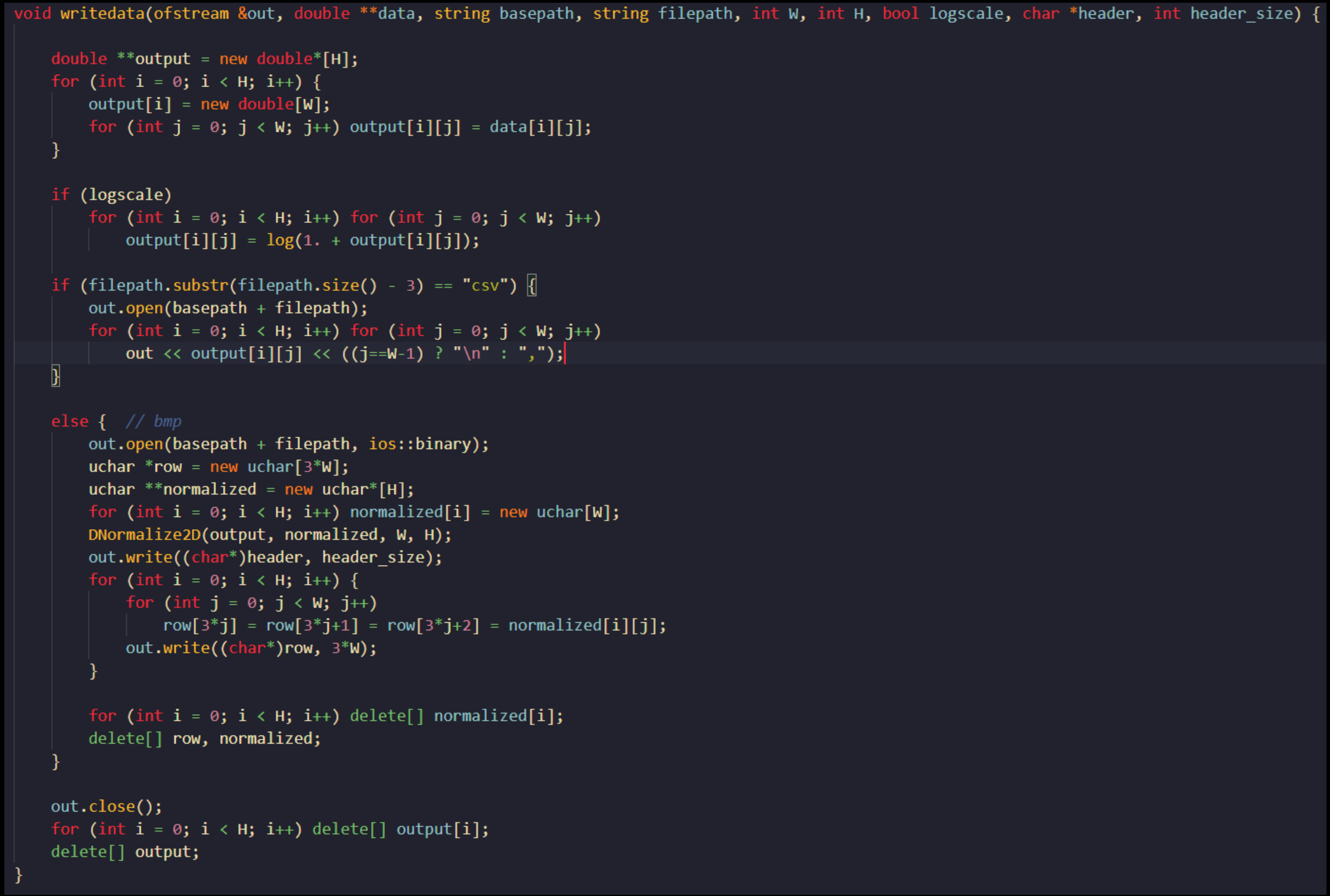
Butter worth filter



```
void filter(int ord, double* a, double* b, int np, double* x, double* y)
{
    int i, j;
    y[0] = b[0] * x[0];
    for (i = 1; i < ord + 1; i++)
    {
        y[i] = 0.0;
        for (j = 0; j < i + 1; j++)
            y[i] = y[i] + b[j] * x[i - j];
        for (j = 0; j < i; j++)
            y[i] = y[i] - a[j + 1] * y[i - j - 1];
    }
    for (i = ord + 1; i < np + 1; i++)
    {
        y[i] = 0.0;
        for (j = 0; j < ord + 1; j++)
            y[i] = y[i] + b[j] * x[i - j];
        for (j = 0; j < ord; j++)
            y[i] = y[i] - a[j + 1] * y[i - j - 1];
    }
}
```



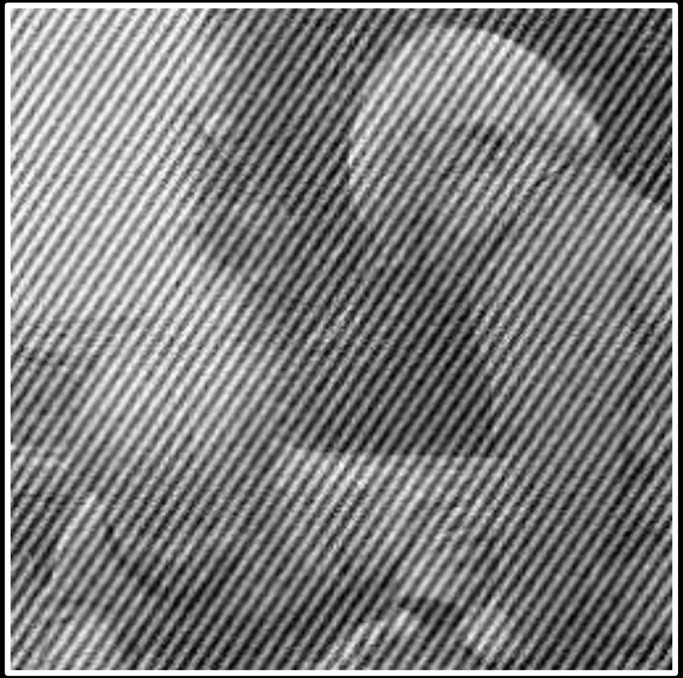
이미지파일 생성



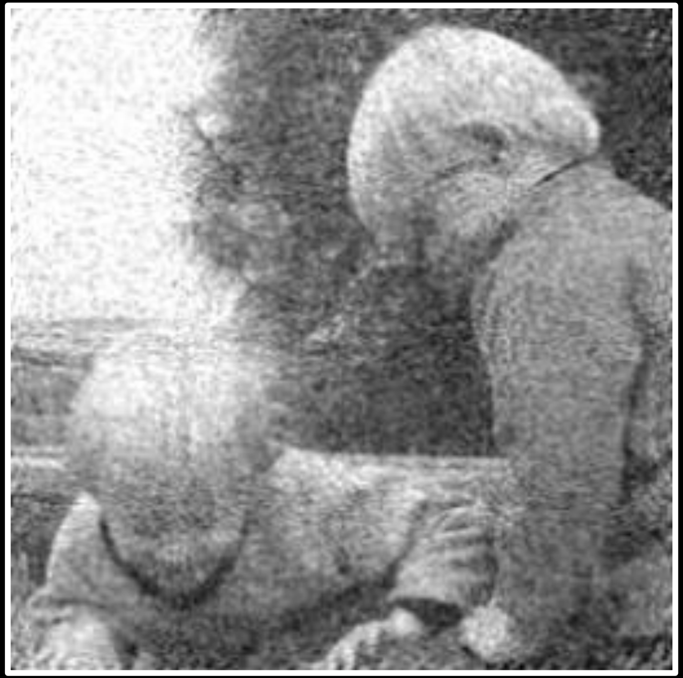
```cpp
void writedata(ofstream &out, double **data, string basepath, string filepath, int W, int H, bool logscale, char *header, int header_size) {

    double **output = new double*[H];
    for (int i = 0; i < H; i++) {
        output[i] = new double[W];
        for (int j = 0; j < W; j++) output[i][j] = data[i][j];
    }

    if (logscale)
        for (int i = 0; i < H; i++) for (int j = 0; j < W; j++)
            output[i][j] = log(1. + output[i][j]);

    if (filepath.substr(filepath.size() - 3) == "csv") {
        out.open(basepath + filepath);
        for (int i = 0; i < H; i++) for (int j = 0; j < W; j++)
            out << output[i][j] << ((j==W-1) ? "\n" : ",");
    }

    else {  // bmp
        out.open(basepath + filepath, ios::binary);
        uchar *row = new uchar[3*W];
        uchar **normalized = new uchar*[H];
        for (int i = 0; i < H; i++) normalized[i] = new uchar[W];
        DNormalize2D(output, normalized, W, H);
        out.write((char*)header, header_size);
        for (int i = 0; i < H; i++) {
            for (int j = 0; j < W; j++)
                row[3*j] = row[3*j+1] = row[3*j+2] = normalized[i][j];
            out.write((char*)row, 3*W);
        }

        for (int i = 0; i < H; i++) delete[] normalized[i];
        delete[] row, normalized;
    }

    out.close();
    for (int i = 0; i < H; i++) delete[] output[i];
    delete[] output;
}
```

3. 결과 분석

1. 노이즈 이미지 vs 복원 이미지



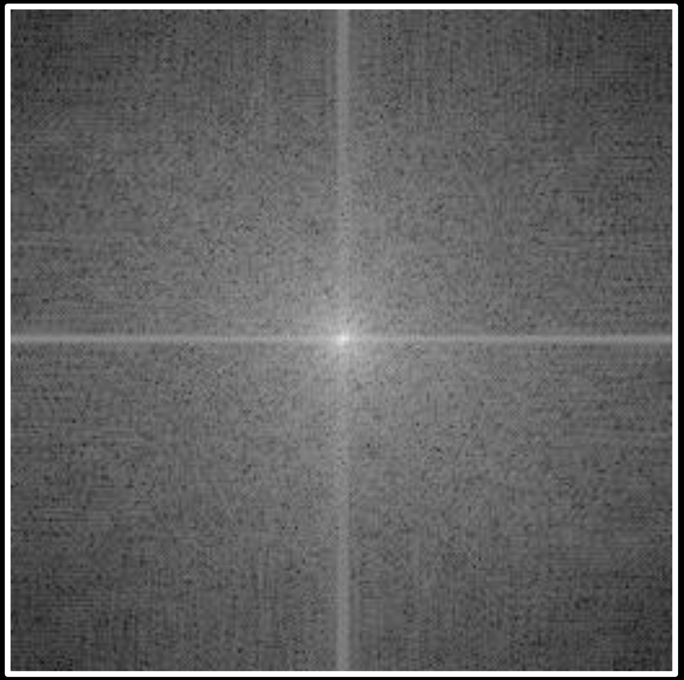
노이즈 이미지



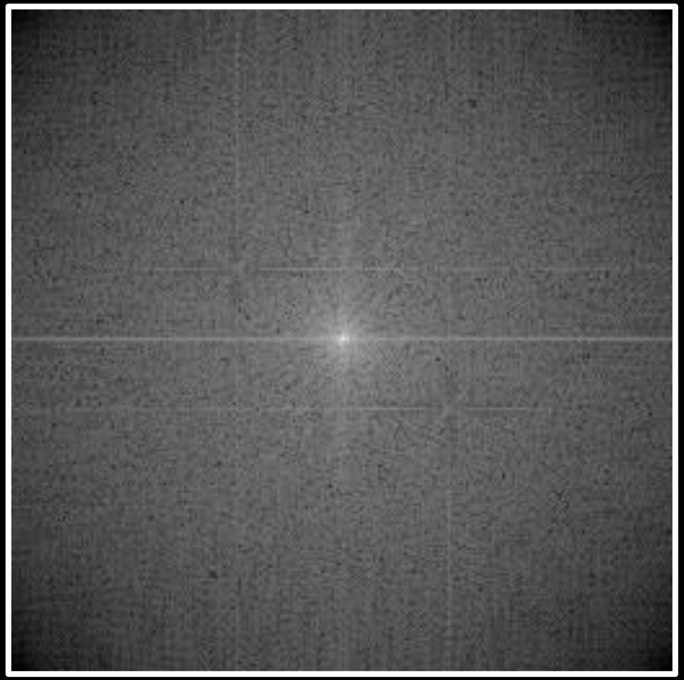
복원 이미지

3. 결과 분석

2. 원본 스펙트럼 vs 복원 스펙트럼



VS

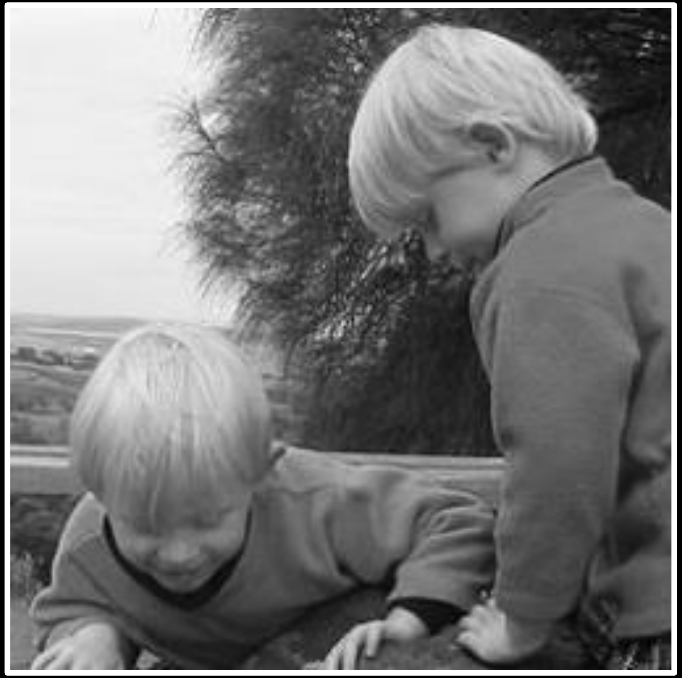


원본 스펙트럼

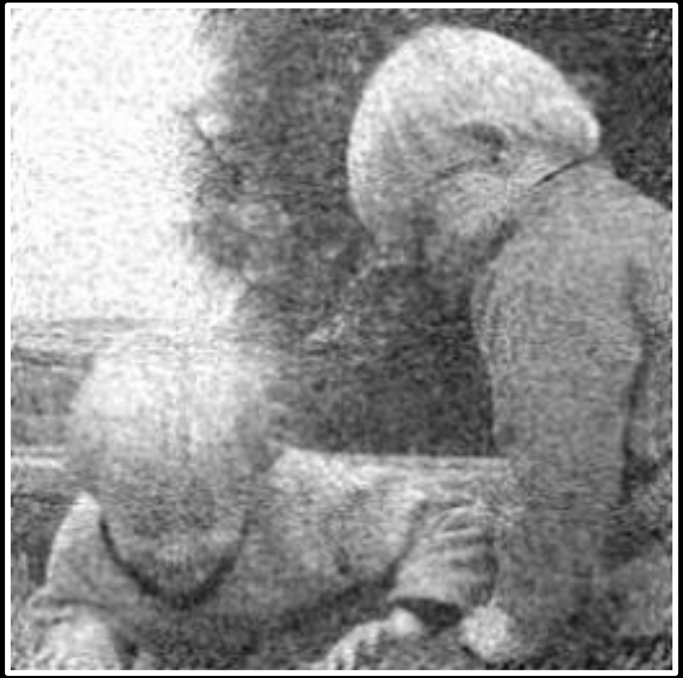
복원 스펙트럼

3. 결과 분석

3. 원본 이미지 vs 복원 이미지


원본 이미지

VS


복원 이미지

4. 결론

5. Q&A

무엇이든
물어보세요~~~~~~