

# DSP 실험

## 프로젝트 발표

---

2019.12.05

**DSP 실험**

김규헌 교수님

**3조**

2015103961 강현경

2015104124 진우빈

2012104053 허수경

2015104045 서재하

2015104127 최원혁



# INDEX



## Part 01.

Zero Padding  
252X252 → 256X256



## Part 02.

2D FFT  
Normalization  
Centralization



## Part 03.

Noise 제거  
Inverse 2D FFT



## Part 04.

Masking  
최종 이미지 출력



**01**

Zero Padding



# 라이브러리 & 매크로 & 사용 함수

```
1 #include <iostream>
2 #include <cmath>
3 #include <fstream>
4 #include <cstdlib>
5 #include "complex.h"
6 using namespace std;
7 #define PHI 3.141592
8 #define W 256
9 #define H 256
10
11 #define W1 252
12 #define H1 252
13 #define uchar unsigned char
14 #define Datalength 256
15 #define WORD unsigned short
16 #define DWORD unsigned int
17
18 void FFT2Radix(double* Xr, double* Xi, double* Yr, double* Yi, int nN, int binverse);
19 void FFT2D(uchar** img, double** OutputReal, double** OutputImag, int nw, int nh);
20 void FFT2D_inverse(double** InputReal, double** InputImag, double** OutputDouble, int nw, int nh);
21 void DNormalize2D(double** p1, uchar** p2, int nw, int nh);
```

>> 라이브러리

>> #define 매크로

>> 사용 함수

# KHU 01. Zero Padding

```
23 int main() {
24
25     //노이즈 있는 252크기 파일//
26     ifstream infile;
27     infile.open("twins_noise.bmp", ios::binary);
28     //256 크기 파일//
29     ifstream infile2;
30     infile2.open("twins_256.bmp", ios::binary);
31
32
33
34
35     char* header;
36     char* header2;
37
38
39
40
41
42
43
44
45
46
47
48     header2 = new char[54];
49     header = new char[54];
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 infile.read((char*)header, 54);
101 infile2.read((char*)header2, 54);
102
103
104 //헤더정보 출력 252 크기 일 때, 256 크기 일 때 각각
105 cout << "<252X252 bmp 파일 헤더 정보>" << endl;
106 cout << "매직넘버" << " " << " " << header[0] << header[1] << endl;
107 cout << "BMP파일 전체크기" << " " << " " << *(DWORD*)(header + 2) << endl;
108 cout << "OFFSET" << " " << " " << *(DWORD*)(header + 10) << endl;
109
110
111 cout << "가로화소" << " " << " " << *(DWORD*)(header + 18) << endl;
112 cout << "세로화소" << " " << " " << *(DWORD*)(header + 22) << endl;
113 cout << "데이터 총 크기" << " " << " " << *(DWORD*)(header + 34) << endl;
114 cout << "가로 해상도" << " " << " " << *(DWORD*)(header + 38) << endl;
115 cout << "세로 해상도" << " " << " " << *(DWORD*)(header + 42) << endl;
116 cout << endl << endl;
117
118 cout << "<256X256 bmp 파일 헤더 정보>" << endl;
119 cout << "매직넘버" << " " << " " << header2[0] << header2[1] << endl;
120 cout << "BMP파일 전체크기" << " " << " " << *(DWORD*)(header2 + 2) << endl;
121 cout << "OFFSET" << " " << " " << *(DWORD*)(header2 + 10) << endl;
122
123
124 cout << "가로화소" << " " << " " << *(DWORD*)(header2 + 18) << endl;
125 cout << "세로화소" << " " << " " << *(DWORD*)(header2 + 22) << endl;
126 cout << "데이터 총 크기" << " " << " " << *(DWORD*)(header2 + 34) << endl;
127 cout << "가로 해상도" << " " << " " << *(DWORD*)(header2 + 38) << endl;
128 cout << "세로 해상도" << " " << " " << *(DWORD*)(header2 + 42) << endl;
129 cout << endl << endl;
```

>> 파일 열기, Header 설정

>> Header 정보 추출

```
<252X252 bmp 파일 헤더 정보>
매직넘버 BM
BMP파일 전체크기 190566
OFFSET 54
가로화소 252
세로화소 252
데이터 총 크기 190512
가로 해상도 3937
세로 해상도 4213
```

```
<256X256 bmp 파일 헤더 정보>
매직넘버 BM
BMP파일 전체크기 196662
OFFSET 54
가로화소 256
세로화소 256
데이터 총 크기 196608
가로 해상도 0
세로 해상도 0
```

계속하려면 아무 키나 누르십시오 .

Header 정보

# KHU 01. Zero Padding

```
129 //노이즈 있는 파일 불러들여오기
130 for (int i = 0; i < H1; i++)
131     infile.read((char*)RGB[i], 3 * W1);
132
133 for (int i = 0; i < H1; i++) {
134     for (int j = 0, jj = 0; j < W1; j++, jj += 3) {
135         R[i][j] = RGB[i][jj];
136     }
137 }
138
139 //초기화
140 for (int i = 0; i < H; i++) {
141     for (int j = 0; j < W; j++) {
142         R256[i][j] = 0;
143     }
144 }
145
146 //FFT를 하기 위해 252크기를 256크기 배열로 옮겨준다
147 for (int i = 0; i < H1; i++) {
148     for (int j = 0; j < W1; j++) {
149         R256[i][j] = R[i][j];
150     }
151 }
152
153 for (int i = 0; i < H; i++) {
154     for (int j = 0, jj = 0; j < W; j++, jj += 3) {
155         RGB256[i][jj] = R256[i][j];
156         RGB256[i][jj + 1] = R256[i][j];
157         RGB256[i][jj + 2] = R256[i][j];
158     }
159 }
160
161
162
163
164
165
166
167
168
169
170
171
172
173 ofstream Outfile0;
174 Outfile0.open("Zero_Padding.bmp", ios::binary);
175 Outfile0.write((char*)header2, 54);
176 for (int i = 0; i < H; i++)
177     Outfile0.write((char*)RGB256[i], 3 * W);
178 Outfile0.close();
```

>> 252X252 → 256X256



Zero\_Padding

크기:

192KB (196,662 바이트)

= 54 + 256 X 256 X 3

디스크 할당 크기:

196KB (200,704 바이트)



Zero Padding (256X256)

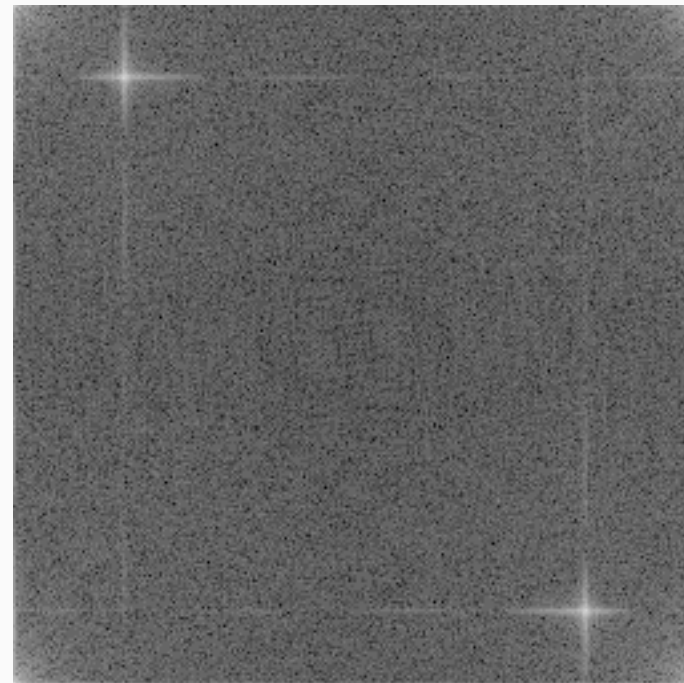


**02**

2D FFT

Normalization & Centralization

```
181 //FFT 실행//
182 FFT2D(R256, fftRe, fftIm, W, H);
183
184 >> 2D FFT
185
186 for (int i = 0; i < H; i++) {
187     for (int j = 0; j < W; j++) {
188         fft[i][j] = complex(fftRe[i][j], fftIm[i][j]);
189         mag[i][j] = 10 * log(fft[i][j].mag() + 1);
190     }
191 }
192
193 // 정규화
194 DNormalize2D(mag, R_, W, H);
195
196 for (int i = 0; i < H; i++) {
197     for (int j = 0, jj = 0; j < W; j++, jj += 3) {
198         RGB_[i][jj] = R_[i][j];
199         RGB_[i][jj + 1] = R_[i][j];
200         RGB_[i][jj + 2] = R_[i][j];
201     }
202 }
203
204 >> Normalization
205
206 // 저주파, 고주파수 대역 보기 위해 출력
207 ofstream Outfile;
208 Outfile.open("Normalization_result.bmp", ios::binary);
209 Outfile.write((char*)header2, 54);
210 for (int i = 0; i < H; i++)
211     Outfile.write((char*)RGB_[i], 3 * W);
212 Outfile.close();
```



Normalization

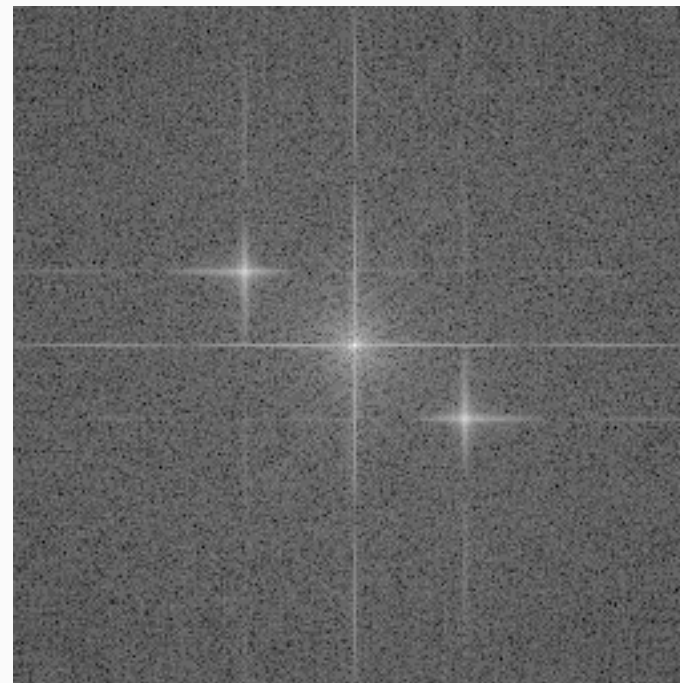


# KHU 02. Centralization

```
213 //주파수 정렬
214 const int center_i = 128;
215 const int center_j = 128;
216
217 for (int i = center_i; i < W; i++) {
218     for (int j = center_j; j < H; j++) {
219         R_Freq_sort[i - center_i][j - center_j] = R_[i][j];
220         //4사분면을 2사분면으로
221     }
222 }
223
224 for (int i = 0; i < center_i; i++) {
225     for (int j = 0; j < center_j; j++) {
226         R_Freq_sort[i + center_i][j + center_j] = R_[i][j];
227         //2사분면을 4사분면으로
228     }
229 }
230
231 for (int i = 0; i < center_i; i++) {
232     for (int j = center_j; j < H; j++) {
233         R_Freq_sort[i + center_i][j - center_j] = R_[i][j];
234         //1사분면을 3사분면으로
235     }
236 }
237
238 for (int i = center_i; i < W; i++) {
239     for (int j = 0; j < center_j; j++) {
240         R_Freq_sort[i - center_i][j + center_j] = R_[i][j];
241         //3사분면을 1사분면으로
242     }
243 }
244
245 for (int i = 0; i < H; i++) {
246     for (int j = 0; jj = 0; j < W; j++, jj += 3) {
247         RGB_Freq_sort[i][jj] = R_Freq_sort[i][j];
248         RGB_Freq_sort[i][jj + 1] = R_Freq_sort[i][j];
249         RGB_Freq_sort[i][jj + 2] = R_Freq_sort[i][j];
250     }
251 }
252 }
```

>> Centralization 결과 출력

```
254 //주파수 정렬 후 출력//
255 ofstream Outfile2;
256 Outfile2.open("Freq_sort_result.bmp", ios::binary);
257 Outfile2.write((char*)header2, 54);
258 for (int i = 0; i < H; i++)
259     Outfile2.write((char*)RGB_Freq_sort[i], 3 * W);
260 Outfile2.close();
```



Centralization



**03**

Noise 제거

# KHU 03. Noise 위치 확인

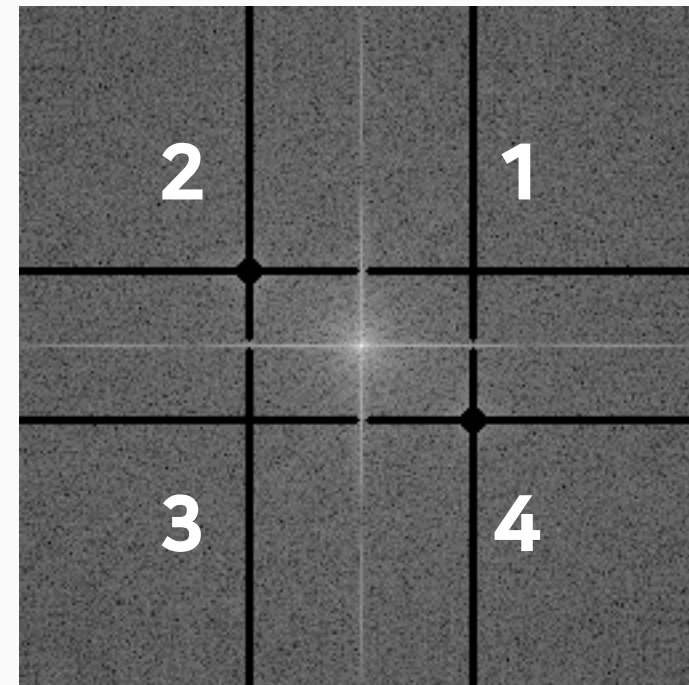
```
263 // 노이즈 위치 확인
264 for (int i = 0; i < H; i++)
265 {
266     for (int j = 0, jj = 0; j < W; j++, jj += 3)
267     {
268         for (int k = 0; k < 127; k += 1)
269         {
270             if
271             ((i - 100) * (i - 100) + (j - k) * (j - k) < 2)
272                 R_Freq_sort[i][j] = 0;
273             >> 4사분면 세로
274         }
275         for (int k = 131; k < 256; k += 1)
276         {
277             if
278             ((i - 100) * (i - 100) + (j - k) * (j - k) < 2)
279                 R_Freq_sort[i][j] = 0;
280             >> 1사분면 세로
281         }
282         for (int k = 0; k < 127; k += 1)
283         {
284             if
285             ((i - k) * (i - k) + (j - 170) * (j - 170) < 2)
286                 R_Freq_sort[i][j] = 0;
287             >> 4사분면 가로
288         }
289         for (int k = 131; k < 256; k += 1)
290         {
291             if
292             ((i - k) * (i - k) + (j - 170) * (j - 170) < 2)
293                 R_Freq_sort[i][j] = 0;
294             >> 3사분면 가로
```

```
295
296
297     for (int k = 0; k < 127; k += 1)
298     {
299         if
300         ((i - 156) * (i - 156) + (j - k) * (j - k) < 2)
301             R_Freq_sort[i][j] = 0;
302         >> 2사분면 세로
303     }
304     for (int k = 131; k < 256; k += 1)
305     {
306         if
307         ((i - 156) * (i - 156) + (j - k) * (j - k) < 2)
308             R_Freq_sort[i][j] = 0;
309         >> 3사분면 세로
310     }
311     for (int k = 0; k < 127; k += 1)
312     {
313         if
314         ((i - k) * (i - k) + (j - 86) * (j - 86) < 2)
315             R_Freq_sort[i][j] = 0;
316         >> 2사분면 가로
317     }
318     for (int k = 131; k < 256; k += 1)
319     {
320         if
321         ((i - k) * (i - k) + (j - 86) * (j - 86) < 2)
322             R_Freq_sort[i][j] = 0;
323         >> 1사분면 가로
324     }
325
326     if
327     ((i - 100) * (i - 100) + (j - 170) * (j - 170) < 25)
328         R_Freq_sort[i][j] = 0;
329     >> 2사분면 점
330
331     // 오른쪽
332     else if
333     ((i - 156) * (i - 156) + (j - 86) * (j - 86) < 25)
334         R_Freq_sort[i][j] = 0;
335     >> 4사분면 점
```

# KHU 03. Noise 위치 확인

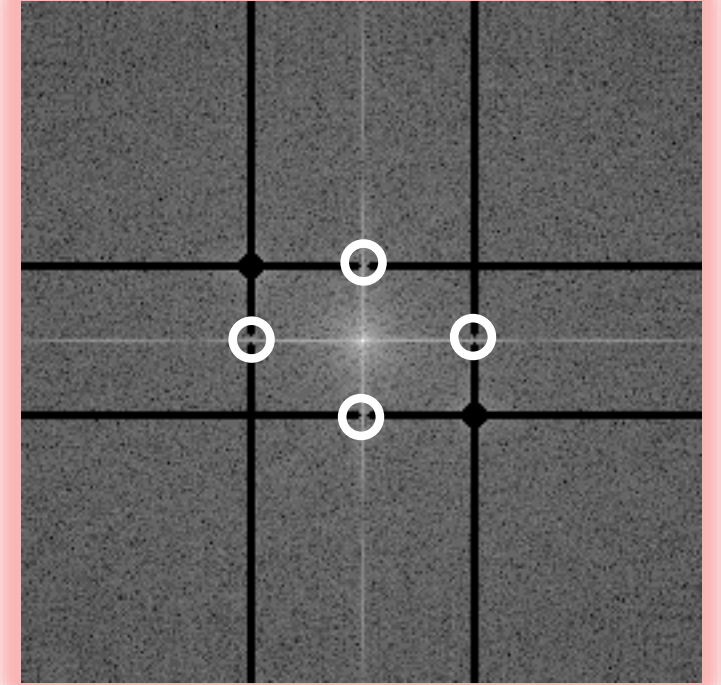
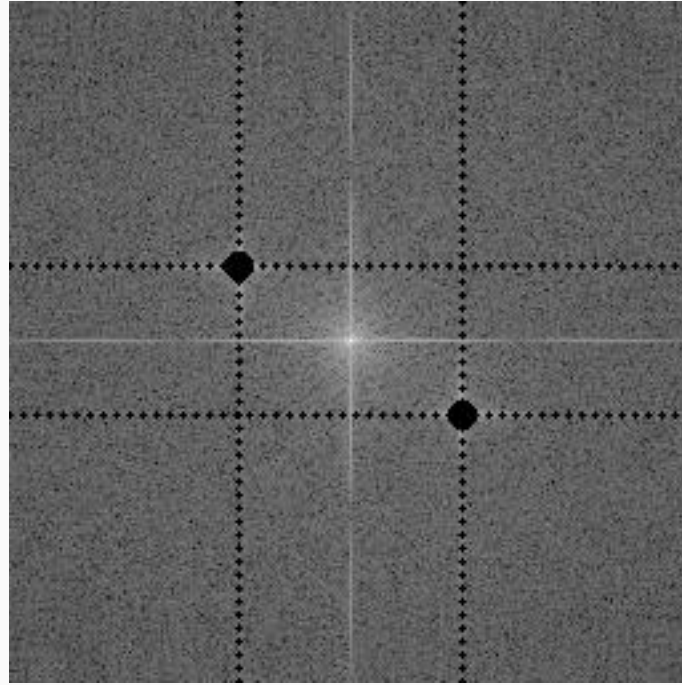
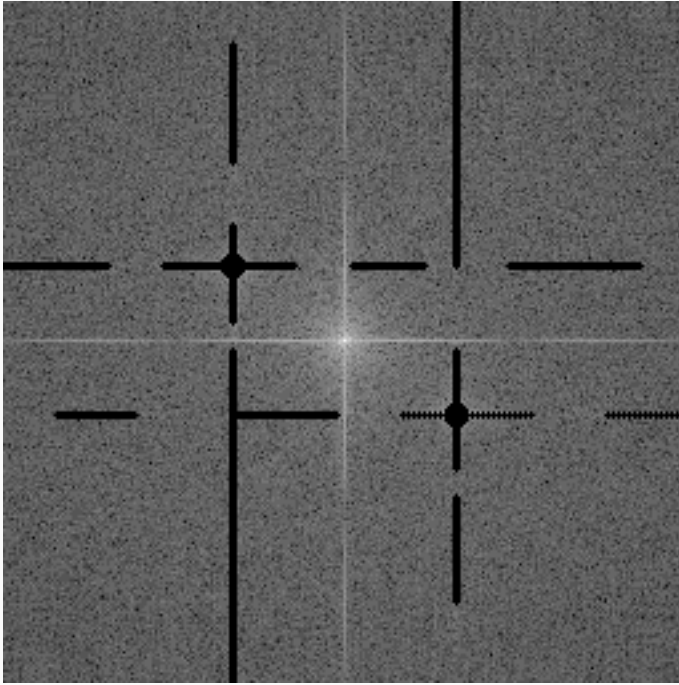
```
331     RGB_Freq_sort[i][jj] = R_Freq_sort[i][j];
332     RGB_Freq_sort[i][jj + 1] = R_Freq_sort[i][j];
333     RGB_Freq_sort[i][jj + 2] = R_Freq_sort[i][j];
334
335 }
336
337
338
339 ofstream Outfile3;
340 Outfile3.open("Freq_sort_Noise_Location.bmp", ios::binary);
341 Outfile3.write((char*)header2, 54);
342 for (int i = 0; i < H; i++)
343     Outfile3.write((char*)RGB_Freq_sort[i], 3 * W);
344 Outfile3.close();
```

>> Noise 위치 확인 결과 출력



Noise 위치 확인

## KHU 03. Noise 위치 확인



시행착오 후 Noise 가장 효과적으로 제거되는 것 선택

```

347 //노이즈 확인 후 제거 시작
348
349
350 // FFT Centralization -> Noise 좌표 알아낸대로 필터링하기 위하여
351 ///////////////////////////////////////////////////
352 for (int i = 0; i < H / 2; i++)
353 {
354     for (int j = 0; j < W / 2; j++) {
355
356         temp2[i][j] = fft[i][j];
357         fft[i][j] = fft[i + (H / 2)][j + (W / 2)];
358         fft[i + (H / 2)][j + (W / 2)] = temp2[i][j];
359     }
360 }
361
362 for (int i = 0; i < H / 2; i++)
363 {
364     for (int j = W / 2; j < W; j++)
365     {
366         temp2[i][j] = fft[i][j];
367         fft[i][j] = fft[i + (H / 2)][j - (W / 2)];
368         fft[i + (H / 2)][j - (W / 2)] = temp2[i][j];
369     }
370 }

```

&gt;&gt; FFT 결과 Centralization

```

372 //FFT noise 필터링
373 ///////////////////////////////////////////////////
374
375 for (int i = 0; i < H; i++)
376 {
377     for (int j = 0, jj = 0; j < W; j++, jj += 3)
378     {
379         for (int k = 0; k < 127; k += 1)
380         {
381             if
382                 ((i - 100) * (i - 100) + (j - k) * (j - k) < 2)
383                 fft[i][j] = complex(0, 0);
384         }
385         for (int k = 130; k < 256; k += 1)
386         {
387             if
388                 ((i - 100) * (i - 100) + (j - k) * (j - k) < 2)
389                 fft[i][j] = complex(0, 0);
390         }
391
392         for (int k = 0; k < 126; k += 1)
393         {
394             if
395                 ((i - k) * (i - k) + (j - 170) * (j - 170) < 2)
396                 fft[i][j] = complex(0, 0);
397         }
398         for (int k = 130; k < 256; k += 1)
399         {
400             if
401                 ((i - k) * (i - k) + (j - 170) * (j - 170) < 2)
402                 fft[i][j] = complex(0, 0);
403         }

```

&gt;&gt; FFT 결과 Noise 제거

# KHU 03. Noise 제거

```
443 // FFT Centralization 재정렬 -> 원래대로 돌려놓기
444 ///////////////////////////////////////////////////
445 for (int i = 0; i < H / 2; i++)
446 {
447     for (int j = 0; j < W / 2; j++) {
448         temp2[i][j] = fft[i][j];
449         fft[i][j] = fft[i + (H / 2)][j + (W / 2)];
450         fft[i + (H / 2)][j + (W / 2)] = temp2[i][j];
451     }
452 }
453
454
455 for (int i = 0; i < H / 2; i++)
456 {
457     for (int j = W / 2; j < W; j++) {
458         temp2[i][j] = fft[i][j];
459         fft[i][j] = fft[i + (H / 2)][j - (W / 2)];
460         fft[i + (H / 2)][j - (W / 2)] = temp2[i][j];
461     }
462 }
463
464 ///////////////////////////////////////////////////
465
466 //fftRe, fftIm 초기화
467 for (int i = 0; i < H; i++) {
468     for (int j = 0; j < W; j++) {
469         fftRe[i][j] = fft[i][j].re;
470         fftIm[i][j] = fft[i][j].im;
471     }
472 }
473
474
```

>> FFT Centralization 결과 재정렬

>> fftRe, fftIm 초기화

```
480 //Inverse FFT 진행//
481 double** result_data;
482 result_data = new double* [H];
483 for (int i = 0; i < H; i++)
484     result_data[i] = new double[W];
485
486 FFT2D_inverse(fftRe, fftIm, result_data, W, H);
487
488
489
490 double** result_data_252;
491 result_data_252 = new double* [H1];
492 for (int i = 0; i < H1; i++)
493     result_data_252[i] = new double[W1];
494
495
496 for (int i = 0; i < H1; i++) {
497     for (int j = 0; j < W1; j++) {
498         result_data_252[i][j] = result_data[i][j];
499     }
500 }
501
502 uchar** noise_eli;
503 noise_eli = new uchar * [H1];
504 for (int i = 0; i < H1; i++)
505     noise_eli[i] = new uchar[3 * W1];
506
507
508 for (int i = 0; i < H1; i++) {
509     for (int j = 0, jj = 0; j < W1; j++, jj += 3) {
510         noise_eli[i][jj] = result_data_252[i][j];
511         noise_eli[i][jj + 1] = result_data_252[i][j];
512         noise_eli[i][jj + 2] = result_data_252[i][j];
513     }
514 }
```


>> Inverse 2D FFT

>> RGB 합치기




```
517 ofstream fftinverse;  
518 fftinverse.open("noise_elminated_but_notmask.bmp", ios::binary);  
519 fftinverse.write((char*)header, 54);  
520  
521  
522 for (int i = 0; i < H1; i++)  
523     fftinverse.write((char*)noise_eli[i], 3 * W1);
```

>> Noise 제거 결과 출력

 noise\_elminated\_but\_notmask

---

파일 형식: 알씨 BMP 파일(.bmp)

연결 프로그램:  ALSee (데스크톱) 변경(C)...

---

위치: C:\Users\User\Desktop\Final\Final

크기: 186KB (190,566 바이트) =  $54 + 252 \times 252 \times 3$

디스크 할당 크기: 188KB (192,512 바이트)



Noise 제거 결과





**04**

Masking

# KHU 04. Masking

```
451 //마스킹 작업 시작//
452 uchar* Orig;
453 Orig = new uchar[252 * 252];
454
455 for (int i = 0; i < H1; i++) {
456     for (int j = 0; j < W1; j++) {
457         Orig[i * 252 + j] = R252_clean[i][j];
458     }
459 }
460
461 double* temp;
462 temp = new double[252 * 252];
463
464 //초기화
465 for (int i = 0; i < 252 * 252; i++) {
466     temp[i] = 0;
467 }
468
469 int m, n;
470 double norm = 0;
471
472 double kernel[3][3] = { {-1, -1, -1},
473     {-1, 9, -1},
474     {-1, -1, -1} };
475
```

```
476 for (m = 0; m < 3; m++) {
477     for (n = 0; n < 3; n++) {
478         norm += kernel[m][n];
479     }
480 }
481 if (norm == 0)
482     norm = 1;
483 else
484     norm = 1 / norm;
```

```
487 for (int row = 1; row < H1 - 1; row++) {
488     for (int col = 1; col < W1 - 1; col++) {
489         for (m = 0; m < 3; m++) {
490             for (n = 0; n < 3; n++) {
491                 temp[row * 252 + col] += (kernel[m][n] * (double)Orig[(row - 1 + m) * 252 + (col - 1 + n)]) * norm;
492             }
493         }
494     }
495 }
```

>> masking 필터 구현

# KHU 04. Masking

```
575 // 마스크 작업 후 값이 정해지지 않은 외곽부분 메우기
576 for (int row = 0; row < H1; row++) {
577     for (int col = 0; col < W1; col++) {
578         switch (row) {
579             case 0:
580                 temp[row * W1 + col] = temp[(row + 1) * W1 + col];
581                 break;
582             case 251:
583                 temp[row * W1 + col] = temp[(row - 1) * W1 + col];
584                 break;
585         }
586     }
587 }
588 for (int row = 0; row < H1; row++) {
589     for (int col = 0; col < W1; col++) {
590         switch (col) {
591             case 0:
592                 temp[row * W1 + col] = temp[row * W1 + col + 1];
593                 break;
594             case 251:
595                 temp[row * W1 + col] = temp[row * W1 + col - 1];
596                 break;
597         }
598     }
599 }
```

>> 외곽 부분 메우기

```
601 //윤곽검출//
602 for (int Row = 0; Row < H1; Row++) {
603     for (int Col = 0; Col < W1; Col++) {
604         temp[Row * W1 + Col] = temp[Row * W1 + Col] - Orig[Row * W1 + Col];
605     }
606 }
607
608 //최대최소//
609 for (int row = 0; row < H1; row++) {
610     for (int col = 0; col < W1; col++) {
611         if (temp[row * W1 + col] < 0) temp[row * W1 + col] = 0;
612         else if (temp[row * W1 + col] > 255) temp[row * W1 + col] = 255;
613     }
614 }
615
616 //마스킹 출력//
617 uchar** masking;
618 masking = new uchar * [H1];
619 for (int i = 0; i < H1; i++)
620     masking[i] = new uchar[3 * W1];
621
622 for (int i = 0; i < H1; i++) {
623     for (int j = 0, jj = 0; j < W1; j++, jj += 3) {
624         masking[i][jj] = temp[252 * i + j];
625         masking[i][jj + 1] = temp[252 * i + j];
626         masking[i][jj + 2] = temp[252 * i + j];
627     }
628 }
629 }
```

>> 윤곽 검출

>> 최대 최소

>> masking 이미지 파일 출력

# KHU 04. Masking

```
631 ofstream mask;
632 mask.open("masking.bmp", ios::binary);
633 mask.write((char*)header, 54);
634
635
636 for (int i = 0; i < H1; i++)
637     mask.write((char*)masking[i], 3 * W1);
```

>> Masking 이미지 결과 출력



masking

파일 형식:

알씨 BMP 파일(.bmp)

연결 프로그램:



ALSee (데스크톱)

변경(C)...

위치:

C:\Users\User\Desktop\Final\Final

크기:

186KB (190,566 바이트)

디스크 할당 크기:

188KB (192,512 바이트)



Masking

# KHU 04. Masking

```
639 //noise 제거 된 파일과 합치기//
640 for (int i = 0; i < H1; i++) {
641     for (int j = 0; j < W1; j++) {
642         result_data_252[i][j] = result_data_252[i][j] + 0.3 * temp[252 * i + j];
643     }
644 }
645
646 for (int row = 0; row < H1; row++) {
647     for (int col = 0; col < W1; col++) {
648         if (result_data_252[row][col] < 0) result_data_252[row][col] = 0;
649         else if (result_data_252[row][col] > 255) result_data_252[row][col] = 255;
650     }
651 }
652
653 uchar** end_output;
654 end_output = new uchar * [H1];
655 for (int i = 0; i < H1; i++)
656     end_output[i] = new uchar[3 * W1];
657
658
659 for (int i = 0; i < H1; i++) {
660     for (int j = 0, jj = 0; j < W1; j++, jj += 3) {
661         end_output[i][jj] = result_data_252[i][j];
662         end_output[i][jj + 1] = result_data_252[i][j];
663         end_output[i][jj + 2] = result_data_252[i][j];
664     }
665 }
666
```

>> masking 합치기

```
670 ofstream Final1;
671 Final1.open("Final_noise_eliminated_output.bmp", ios::binary);
672 Final1.write((char*)header, 54);
673
674
675 for (int i = 0; i < H1; i++)
676     Final1.write((char*)end_output[i], 3 * W1);
```

>> 최종 이미지 출력



Final\_noise\_eliminated\_output

크기: 186KB (190,566 바이트)

디스크 할당 크기: 188KB (192,512 바이트)



최종 이미지 파일 (252X252)

# KHU 04. Masking



+



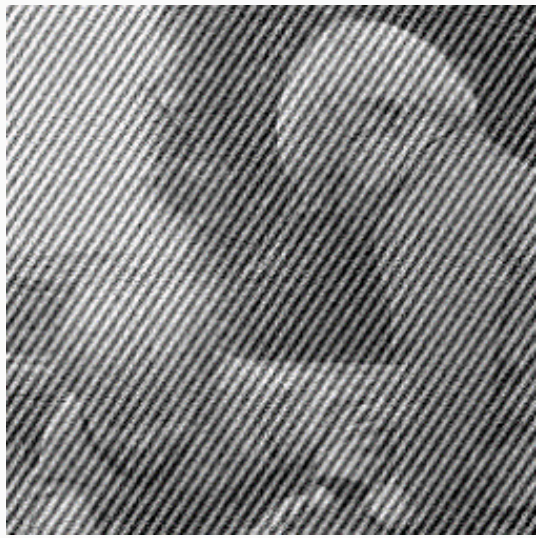
2D FFT, Noise 제거

Masking

최종 이미지



# KHU Noise 제거 과정



Noise 이미지 파일

Zero Padding

Noise 제거

Sharpening

「**감사합니다.**」

**Q & A**