```
#include <stdio.h>
#include <process.h>
#include <Windows.h>
#include <string.h>
#include <tchar.h>
#include <stdbool.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#pragma warning(disable : 4996)
// 시간을 계산하기 위한 변수
double total_Time_GPU, total_Time_CPU, tmp_time;
LARGE_INTEGER beginClock, endClock, clockFreq;
LARGE_INTEGER tot_beginClock, tot_endClock, tot_clockFreq;

// 비트맵 이미지 정보를 받아오기 위한 구조체 변수
BITMAPFILEHEADER bfh;
BITMAPINFOHEADER bih;
RGBQUAD * rgb;

// 이미지 정보를 다루기 위해 사용하는 변수
int bpl, bph;
unsigned char* pix; // 원본 이미지
unsigned char* pix_hvs; // 하프톤 이미지


double pi = 3.14159265358979323846264338327950288419716939937510;

double G[7][7];
int fs = 7;       // 가우시안 필터 사이즈
double CPP[13][13]; // 자기상관 행렬(G, G)
int halfcppsize = 6;

/*
double G[11][11];
int fs = 11;      // 가우시안 필터 사이즈
double CPP[21][21];
int halfcppsize = 10;
*/
```

```
double* CEP;     // 상호상관 행렬(err, CPP)
char str[100];                              // 파일명을 담을 문자열

void DBS();                                          // Direct Binary Search 연산
void GaussianFilter();              // 가우시안 필터 생성
double GaussianRandom(double stddev, double average);                  //   정규분포
난수 생성
void CONV();    // 2차원 컨볼루션 연산
void XCORR();  // 상호상관관계 연산
void Halftone();                            // 초기 하프톤 이미지 생성
void FwriteCPU(char *);          // 연산된 픽셀값을 bmp파일로 저장하는 함수

int main(void)
{
        FILE * fp;
        //fp = fopen("EDIMAGE.bmp", "rb");
        //fp = fopen("newEDIMAGE2.bmp", "rb");
        fp = fopen("test.bmp", "rb");
        //fp = fopen("bird_K.bmp", "rb");

        fread(&bfh, sizeof(bfh), 1, fp);
        fread(&bih, sizeof(bih), 1, fp);

        rgb = (RGBQUAD*)malloc(sizeof(RGBQUAD) * 256);
        fread(rgb, sizeof(RGBQUAD), 256, fp);

        // BPL을 맞춰주기 위해서 픽셀데이터의 사이즈를 4의 배수로 조정
        bpl = (bih.biWidth + 3) / 4 * 4;
        bph = (bih.biHeight + 3) / 4 * 4;

        pix = (unsigned char *)malloc(sizeof(unsigned char) * bpl * bph);
        memset(pix, 0, sizeof(unsigned char) * bpl * bph);
        fread(pix, sizeof(unsigned char), bpl * bph, fp);

        pix_hvs = (unsigned char *)malloc(sizeof(unsigned char) * bpl * bph);
        memset(pix_hvs, 0, sizeof(unsigned char) * bpl * bph);

        CEP = (double *)malloc(sizeof(double) * (bpl + halfcppsize * 2) * (bph +
halfcppsize * 2));
        memset(CEP, 0, sizeof(double) * (bpl + halfcppsize * 2) * (bph + halfcppsize
* 2));
```

```
	QueryPerformanceFrequency(&tot_clockFreq);	// 시간을 측정하기위한 준비

	total_Time_CPU = 0;
	QueryPerformanceCounter(&tot_beginClock); // 시간측정 시작
	// Direct Binary Search
	DBS();
	QueryPerformanceCounter(&tot_endClock);

	total_Time_CPU = (double)(tot_endClock.QuadPart - tot_beginClock.QuadPart)
/ tot_clockFreq.QuadPart;
	printf("Total processing Time_DBS : %f ms\n", total_Time_CPU * 1000);
	//system("pause");

	//sprintf(str, "DBS_Dither.bmp");
	//sprintf(str, "new_DBS_Dither2.bmp");
	sprintf(str, "test_DBS_Dither.bmp");
	//sprintf(str, "bird_DBS_Dither_K.bmp");

	FwriteCPU(str);

	free(rgb);
	free(pix);
	free(pix_hvs);
	free(CEP);
	fclose(fp);

	return 0;
}

void DBS()
{
	int count = 0;			// 최소제곱오차 값이 0이 아닐때까지 반복문을
돌리기위한 카운트 변수
	double eps = 0.0;
	double eps_min = 0.0;
	int a0 = 0;
	int a1 = 0;
	int a0c = 0;
	int a1c = 0;
```

```
int cpx = 0;
int cpy = 0;

Halftone();                                    // 초기 하프톤이미지 생성

GaussianFilter();          // 가우시안 필터 생성

CONV();          // 2차원 컨볼루션 연산 행렬 생성 (CPP)

XCORR();         // 상호관계연산 행렬 생성 (CEP)

// DBS 과정 시작..
while(1)
{
        count = 0;
        a0 = 0;
        a1 = 0;
        a0c = 0;
        a1c = 0;
        cpx = 0;
        cpy = 0;

        for (int i = 1; i < bph - 1; i++)
        {
                for (int j = 1; j < bpl - 1; j++)
                {
                        a0c = 0;
                        a1c = 0;
                        cpx = 0;
                        cpy = 0;
                        eps_min = 0;
                        for (int y = -1; y <= 1; y++)
                        {
                                //if (i + y < 0 || i + y >= bph)
                                        //continue;
                                for (int x = -1; x <= 1; x++)
                                {
                                        //if (j + x < 0 || j + x >= bpl)
                                                //continue;
                                        if (y == 0 && x == 0)
                                        {
```

```
                if (pix_hvs[i * bpl + j] ==
255)
                {
                        a0 = -1;
                        a1 = 0;
                }
                else
                {
                        a0 = 1;
                        a1 = 0;
                }
        }
        else
        {
                if (pix_hvs[(i + y) * bpl + (j
+ x)] != pix_hvs[i * bpl + j])
                {
                        if (pix_hvs[i * bpl +
j] == 255)
                        {
                                a0 = -1;
                                a1 = (-1) *
a0;
                        }
                        else
                        {
                                a0 = 1;
                                a1 = (-1) *
a0;
                        }
                }
                else
                {
                        a0 = 0;
                        a1 = 0;
                }
        }
        eps = (a0 * a0 + a1 * a1) *
CPP[halfcppsize][halfcppsize]
                + 2 * a0 * a1 *
CPP[halfcppsize + y][halfcppsize + x]
```

```
                                                + 2 * a0 * CEP[(i +
halfcppsize) * (bpl + halfcppsize * 2) + (j + halfcppsize)]
                                                + 2 * a1 * CEP[(i + y +
halfcppsize) * (bpl + halfcppsize * 2) + (j + x + halfcppsize)];
                                        if (eps_min > eps)
                                        {
                                                eps_min = eps;
                                                a0c = a0;
                                                a1c = a1;
                                                cpx = x;
                                                cpy = y;
                                        }
                                }
                        }
                        if (eps_min < 0.0)
                        {
                                for (int y = (-1) * halfcppsize; y <=
halfcppsize; y++)
                                {
                                        for (int x = (-1) * halfcppsize; x <=
halfcppsize; x++)
                                        {
                                                CEP[(i + y + halfcppsize) *
(bpl + halfcppsize * 2) + (j + x + halfcppsize)] += (double)a0c * CPP[y +
halfcppsize][x + halfcppsize];
                                                CEP[(i + y + cpy +
halfcppsize) * (bpl + halfcppsize * 2) + (j + x + cpx + halfcppsize)] += (double)a1c *
CPP[y + halfcppsize][x + halfcppsize];
                                        }
                                }
                                pix_hvs[i * bpl + j] += a0c * 255;
                                pix_hvs[(cpy + i) * bpl + (j + cpx)] += a1c *
255;
                                count++;
                        }
                }
        }
        printf("%d\n", count);
        if (count == 0)
                break;
    }
```

```
}
// 가우시안 필터 생성
void GaussianFilter()
{
	double d = 1.2;		// sigma
	double c;
	int gaulen = (fs - 1) / 2;

	for (int k = (-1) * gaulen; k <= gaulen; k++)
	{
		for (int l = (-1) * gaulen; l <= gaulen; l++)
		{
			c = (k * k + l * l) / (2 * d * d);
			G[k + gaulen][l + gaulen] = exp((-1) * c) / (2 * pi * d * d);
			//sum += G[k + gaulen][l + gaulen];


			// fs = 11 일때 결과가 가장 좋았음..
			/*
			c = (k * k + l * l) / (2 * d * d) + 1;
			G[k + gaulen][l + gaulen] = 1 / c;
			*/
		}
	}
}
// 2차원 컨볼루션 연산
void CONV()
{
	double sum = 0;
	for (int y = (-1) * fs + 1; y < fs; y++)
	{
		for (int x = (-1) * fs + 1; x < fs; x++)
		{
			sum = 0;
			for (int i = y; i < y + fs; i++)
			{
				for (int j = x; j < x + fs; j++)
				{
					if ((i >= 0 && j >= 0) && (i < fs && j < fs))
					{
						sum += G[i - y][j - x] * G[i][j];
```

```
                                        }
                                }
                        }
                        CPP[(y + fs - 1)][(x + fs - 1)] = (double)sum;
                }
        }
}
// 상호상관관계 연산
void XCORR()
{
        double sum = 0;
        for (int y = (-1) * halfcppsize * 2; y < bph; y++)
        {
                for (int x = (-1) * halfcppsize * 2; x < bpl; x++)
                {
                        sum = 0;
                        for (int i = y; i <= y + halfcppsize * 2; i++)
                        {
                                for (int j = x; j <= x + halfcppsize * 2; j++)
                                {
                                        if ((i >= 0 && j >= 0) && (i < bph && j <
bpl))
                                        {
                                                // err * CPP
                                                sum += (double)CPP[i - y][j - x] *
((double)pix_hvs[i * bpl + j] / 255 - (double)pix[i * bpl + j] / 255);
                                        }
                                }
                        }
                        CEP[(y + halfcppsize * 2) * (bpl + halfcppsize * 2) + (x +
halfcppsize * 2)] = (double)sum;
                }
        }
}
// 정규분포 난수 발생
double GaussianRandom(double stddev, double average)
{
        double v1, v2, s, temp;

        do {
                v1 = 2 * ((double)rand() / RAND_MAX) - 1;       // -1.0 ~ 1.0 까지
```

```
의 값
                v2 = 2 * ((double)rand() / RAND_MAX) - 1;       // -1.0 ~ 1.0 까지
의 값
                s = v1 * v1 + v2 * v2;
        } while (s >= 1 || s == 0);

        s = sqrt((-2 * log(s)) / s);

        temp = v1 * s;
        temp = (stddev * temp) + average;

        return temp;
}
void Halftone()
{
        // 정규분포 난수로 하프톤이미지 생성

        srand(time(NULL));
        double tmp;
        for (int y = 1; y < bph - 1; y++)
        {
                for (int x = 1; x < bpl - 1; x++)
                {
                        tmp = (double)GaussianRandom(1.0, 0);
                        if (tmp > 0)
                        {
                                pix_hvs[y * bpl + x] = 255;
                        }
                }
        }
}
// 데이터 픽셀값을 bmp파일로 쓴다.
void FwriteCPU(char * fn)
{
        FILE * fp2 = fopen(fn, "wb");
        fwrite(&bfh, sizeof(bfh), 1, fp2);
        fwrite(&bih, sizeof(bih), 1, fp2);
        fwrite(rgb, sizeof(RGBQUAD), 256, fp2);
        fwrite(pix_hvs, sizeof(unsigned char), bpl * bph, fp2);
        fclose(fp2);
}
```