간단정리

DFT

for (int k = 0; k < 16000; k++)

{

for (int n = 0; n < 16000; n++)

{

X[k] += x[n] \* complex(cos((-2 \* PI\*k\*n) / (double)16000), sin((-2 \* PI\*k\*n) / (double)16000));

}

}

IDFT

for (int n = 0; n < 16000; n++)

{

for (int k = 0; k < 16000; k++)

{

x[n] += X[k] \* complex(cos((2 \* PI\*k\*n) / (double)16000), sin((2 \* PI\*k\*n) / (double)16000));

}

x[n] = x[n]/16000;//공식대로

}

DFT2D

for (int v = 0; v < H; v++) //for문 외울것 vuyx~vu

{

for (int u = 0; u < W; u++)

{

for (int y = 0; y < H; y++)

{

for (int x = 0; x < W; x++)

{

R\_[v][u] += complex(R[y][x], 0)\*complex(-2 \* PI\*((double)u\*x / W + (double)v\*y / H));

}

}

}

}

IDFT2D

for (int y = 0; y < H; y++)

{

for (int x = 0; x < W; x++)

{

for (int v = 0; v < H; v++)

{

for (int u = 0; u < W; u++)

{

idft[y][x] += R\_[v][u]\*complex(2 \* PI\*((double)u\*x / W + (double)v\*y / H));

}

}

idft[y][x] = idft[y][x] / (H\*W);

R[y][x] = idft[y][x].re;

}

}

DCT2D

void DCT2D(unchar\*\*R, double\*\*R\_)//변환 결과값:256\*256

{

int N = 8;//한 축당 32개의 이미지로 나눔

int unit = 256 / 8;//이미지 한축 픽셀수/N

double sum = 0;

int v, u, y, x;

for (int mcr\_i = 0; mcr\_i < unit; mcr\_i++)

{

for (int mcr\_j = 0; mcr\_j < unit; mcr\_j++)

{

for (int k = 0; k < N; k++)

{

v = mcr\_i \* N + k;

for (int l; l < N; l++)

{

u = mcr\_j \* N + l;

sum = 0;

for (int i = 0; i < N; i++)

{

y = mcr\_i \* N + i;

for (int j = 0; j < N; j++)

{

x = mcr\_j \* N + j;

double th1 = (double)(2.\*i + 1)\*k\*PI / (2.\*N);

double th2 = (double)(2.\*j + 1)\*l\*PI / (2.\*N);

sum += (double)cos(th1)\*cos(th2)\*R[y][x];

}

}

double ck;

if (k == 0 && l == 0)

ck = 1. / 8;

else if (k != 0 && l != 0)

ck = 1. / 4;

else

ck = sqrt(2.) / 8.;

R\_[v][u] = ck \* sum;

}

}

}

}

}

IDCT2D

void IDCT2D(double\*\* R\_, unchar\*\* R)

{

int N = 8;//한 축당 32개의 이미지로 나눔

int unit = 256 / 8;//이미지 한축 픽셀수/N

double sum = 0;

int v, u, y, x;

for (int mcr\_i = 0; mcr\_i < unit; mcr\_i++)

{

for (int mcr\_j = 0; mcr\_j < unit; mcr\_j++)

{

for (int i = 0; i < N; i++)

{

y = mcr\_i \* N + i;

for (int j = 0; j < N; j++)

{

x = mcr\_j \* N + j;

sum = 0;

for (int k = 0; k < N; k++)

{

v = mcr\_i \* N + k;

for (int l = 0; l < N; l++)

{

u = mcr\_j \* N + l;

double ck;

if (k == 0 && l == 0)

ck = 1. / 8;

else if (k != 0 && l != 0)

ck = 1. / 4;

else

ck = sqrt(2.) / 8.;

sum += ck \* cos((2.\*(double)i + 1)\*k\*PI / 16)\*cos((2.\*(double)j + 1)\*l\*PI / 16)\*R\_[v][u];

}

}

if (sum < 0)

R[y][x] = 0;

else if (sum > 255)

R[y][x] = 255;

else

R[y][x] = (int)sum;

}

}

}

}

}

Normalize2D

void Normalize2D(double \*\*mag, unchar \*\*nR\_, int nW, int nH)

{

int x, y;

double min = 9999.;

double max = -9999.;

double magvalue;

for (y = 0; y < nH; y++)

{

for (x = 0; x < nW; x++)

{

magvalue = mag[y][x];

if (magvalue > max) max = magvalue;

if (magvalue < min) min = magvalue;

}

}

if (max == min)

{

for (y = 0; y < nH; y++)

{

for (x = 0; x < nW; x++)

{

mag[y][x] = 0;

}

}

return;

}

double normalfact = 255 / max - min;

for (y = 0; y < nH; y++)

{

for (x = 0; x < nW; x++)

{

nR\_[y][x] = (unchar)((mag[y][x]-min)\*normalfact);

}

}

}

FFT

void FFT2Radix(double\* Xr, double\* Xi, double\* Yr, double\* Yi, int nN, bool bInverse)

//false : fft, true : ifft

FFT2D

void FFT2D(uchar\*\* img, double\*\* OutputReal, double\*\* OutputImag, int nW, int nH)

{

int x, y;

double \*dRealX, \*dImagX;

double \*dRealY, \*dImagY;

dRealX = new double[nW];

dImagX = new double[nW];

dRealY = new double[nW];

dImagY = new double[nW];

for (y = 0; y < nH; y++) {

for (x = 0; x < nW; x++) {

dRealX[x] = img[y][x];

dImagX[x] = 0.;

}

FFT2Radix(dRealX, dImagX, dRealY, dImagY, nW, false);

for (x = 0; x < nW; x++) {

OutputReal[y][x] = dRealY[x];

OutputImag[y][x] = dImagY[x];

}

}

delete[] dRealX;

delete[] dImagX;

delete[] dRealY;

delete[] dImagY;

dRealX = new double[nH];

dImagX = new double[nH];

dRealY = new double[nH];

dImagY = new double[nH];

for (x = 0; x < nW; x++) {

for (y = 0; y < nH; y++) {

dRealX[y] = OutputReal[y][x];

dImagX[y] = OutputImag[y][x];

}

FFT2Radix(dRealX, dImagX, dRealY, dImagY, nH, false);

for (y = 0; y < nH; y++) {

OutputReal[y][x] = dRealY[y];

OutputImag[y][x] = dImagY[y];

}

}

delete[] dRealX;

delete[] dImagX;

delete[] dRealY;

delete[] dImagY;

}

IFFT2D

void FFT2Dinverse(double\*\* InputReal, double\*\* InputImag, uchar\*\* OutputDouble, int nW, int nH)

{

int x, y;

double \*dRealX, \*dImagX;

double \*dRealY, \*dImagY;

double\*\* OutputReal, \*\*OutputImag;

OutputReal = new double\*[nH];

OutputImag = new double\*[nH];

for (int i = 0; i < nH; i++) {

OutputReal[i] = new double[nW];

OutputImag[i] = new double[nW];

}

dRealX = new double[nW];

dImagX = new double[nW];

dRealY = new double[nW];

dImagY = new double[nW];

for (y = 0; y < nH; y++) {

for (x = 0; x < nW; x++) {

dRealX[x] = InputReal[y][x];

dImagX[x] = InputImag[y][x];

}

FFT2Radix(dRealX, dImagX, dRealY, dImagY, nW, true);

for (x = 0; x < nW; x++) {

OutputReal[y][x] = dRealY[x];

OutputImag[y][x] = dImagY[x];

}

}

delete[] dRealX;

delete[] dImagX;

delete[] dRealY;

delete[] dImagY;

dRealX = new double[nH];

dImagX = new double[nH];

dRealY = new double[nH];

dImagY = new double[nH];

for (x = 0; x < nW; x++) {

for (y = 0; y < nH; y++) {

dRealX[y] = OutputReal[y][x];

dImagX[y] = OutputImag[y][x];

}

FFT2Radix(dRealX, dImagX, dRealY, dImagY, nH, true);

for (y = 0; y < nH; y++) {

OutputReal[y][x] = dRealY[y];

OutputImag[y][x] = dImagY[y];

}

}

delete[] dRealX;

delete[] dImagX;

delete[] dRealY;

delete[] dImagY;

for (y = 0; y < nH; y++) {

for (x = 0; x < nW; x++) {

OutputDouble[y][x] = OutputReal[y][x];

}

}

for (int i = 0; i < nH; i++) {

delete[] OutputReal[i];

delete[] OutputImag[i];

}

delete[] OutputReal;

delete[] OutputImag;

}

Filter

void Filter(int fs, complex\* dft, complex\* output)

{

ofstream out\_mag;

out\_mag.open("mag.txt");

complex\* H = new complex[2 \* fs];

complex\* Z = new complex[2 \* fs];

complex zero1 = complex(2 \* PI \* 0 / 16000)\*0.9;

complex zero2 = complex(2 \* PI \* 8000 / 16000)\*0.9;

complex pole1 = complex(2 \* PI \* 4000 / 16000)\*0.9;

complex pole2 = complex(2 \* PI \* 12000 / 16000)\*0.9;

for (int k = 0; k < 2 \* fs; k++)

{

Z[k] = complex(2 \* PI\*k / (double)(2 \* fs));

H[k] = (Z[k] - zero1)\*(Z[k] - zero2) / ((Z[k] - pole1)\*(Z[k] - pole2));

}

for (int k = 0; k < 2 \* fs; k++)

out\_mag << H[k].mag() << endl;

}