영상처리 HW4

16010811 김범수

DFT_process

```
//DFT
Padding_buf = (UChar*)calloc(BLK * BLK, sizeof(UChar));
for (Int Blk_Row = 0; Blk_Row < img->info.height / BLK; Blk_Row++)
{
    for (Int Blk_COl = 0; Blk_COl < img->info.width / BLK; Blk_COl++)
    {
        memset(Padding_buf, 0, sizeof(UChar) * BLK * BLK);

        Focus_TMP = img->Ori_img + ((Blk_Row * BLK) * img->info.Ori_stride + (Blk_COl * BLK));
        for (int k = 0; k < BLK; k++)
            for (int l = 0; l < BLK; l++)
            {
                TMP = Focus_TMP + (k * img->info.Ori_stride + l);
                Padding_buf[k * BLK + l] = TMP[0];
            }

        DFT_Func(Padding_buf, BLK, Blk_Row, Blk_COl, img);
    }
}
free(Padding_buf);
```

- Padding_buf : BLK * BLK 크기의 메모리 할당
- memset : Padding_buf 값 초기화
- Focus_TMP : Ori_img에서 BLK * BLK 크기 배열의 첫번째 픽셀 주소
- 이후 TMP를 통해 Padding_buf에 픽셀 값 할당
- DFT_Func 진행

```
for (Int Blk_Row = 0; Blk_Row < img->info.height / BLK; Blk_Row++)
{
    for (Int Blk_COl = 0; Blk_COl < img->info.width / BLK; Blk_COl++)
    {
        memset(Mag_buf, 0, sizeof(Double) * BLK * BLK);
        memset(Pha_buf, 0, sizeof(Double) * BLK * BLK);

        Mag_Focus_TMP = img->Magnitude + ((Blk_Row * BLK) * img->info.Ori_stride + (Blk_COl * BLK));
        Pha_Focus_TMP = img->Phase     + ((Blk_Row * BLK) * img->info.Ori_stride + (Blk_COl * BLK));
        for (int k = 0; k < BLK; k++)
            for (int l = 0; l < BLK; l++)
            {
                Mag_TMP = Mag_Focus_TMP + (k * img->info.Ori_stride + l);
                Mag_buf[k * BLK + l] = Mag_TMP[0];

                Pha_TMP = Pha_Focus_TMP + (k * img->info.Ori_stride + l);
                Pha_buf[k * BLK + l] = Pha_TMP[0];
            }

        IDFT_Func(Mag_buf, Pha_buf, BLK, Blk_Row, Blk_COl, img);
    }
}
```

- memset : Mag_buf, Pha_buf 값 초기화
- Mag_Focus_TMP : Magnitude에서 BLK * BLK 크기 배열의 첫번째 픽셀 주소
- Pha_Focus_TMP : Phase에서 BLK * BLK 크기 배열의 첫번째 픽셀 주소
- 이후 TMP를 통해 Mag_buf, Pha_buf 에 픽셀 값 할당
- IDFT_Func 진행

DFT_Func //실수부 허수부 나누기

```
//실수부 허수부 나누기
for (Int i = 0; i < BLK; i++)                                          //블록 내 칸이동
{
    for (Int j = 0; j < BLK; j++)
    {
        for (Int k = 0; k < BLK; k++)
        {
            for (Int l = 0; l < BLK; l++)                              // 크기 위상 구해야함
            {
                DFT_Real[i * BLK + j] += (double)buf[k * BLK + l] * cos(2.0 * PI * (double)(i * k + j * l) / (double)BLK); //실수
                DFT_Imag[i * BLK + j] += (double)buf[k * BLK + l] * sin(2.0 * PI * (double)(i * k + j * l) / (double)BLK);//허수
            }
        }
        DFT_Real[i * BLK + j] = DFT_Real[i * BLK + j] / (BLK * BLK); //실수
        DFT_Imag[i * BLK + j] = DFT_Imag[i * BLK + j] / (BLK * BLK); //허수
    }
}

if (BLK == img->info.width && BLK == img->info.height && FilterFlag == 'y')
    LPF(img, DFT_Real, DFT_Imag, Blk_Row, Blk_Col);
```

- DFT_Real에 Padding_buf와 cos 계수를 곱한 값의 누적 합을 통해 픽셀의 실수 값 저장
- DFT_Imag에 Padding_buf와 sin 계수를 곱한 값의 누적 합을 통해 픽셀의 실수 값 저장

※ 원래 오일러 공식에 따르면 – sin을 곱하는게 맞지만 이후 IDFT 과정에서 (허수 j) x (허수 j)의 과정이 생략되므로 + sin을 곱함

- DFT_Real, DFT_Imag에 BLK*BLK를 나눠줌

DFT_Func // Magnitude, Phase 생성

```
// Magnitude, Phase 생성
for (Int i = 0; i < BLK; i++)
{
    for (Int j = 0; j < BLK; j++)
    {
        Mag_blk[i * BLK + j] = sqrt(DFT_Real[i * BLK + j] * DFT_Real[i * BLK + j] + DFT_Imag[i * BLK + j] * DFT_Imag[i * BLK + j]);
        Pha_blk[i * BLK + j] = atan2(DFT_Imag[i * BLK + j], DFT_Real[i * BLK + j]);
    }
}
```

- 앞에서 구한 DFT_Real, DFT_Imag를 이용해 Magitude, Phase 결정

IDFT_Func

```
for (Int i = 0; i < BLK; i++)
{
    for (Int j = 0; j < BLK; j++)
    {
        DFT_Real[i * BLK + j] = Mag_buf[i * BLK + j] * cos(Pha_buf[i * BLK + j]);
        DFT_Imag[i * BLK + j] = Mag_buf[i * BLK + j] * sin(Pha_buf[i * BLK + j]);
    }
}

for (Int i = 0; i < BLK; i++)
{
    for (Int j = 0; j < BLK; j++)
    {
        Recon_R = 0;
        for (Int k = 0; k < BLK; k++)
        {
            for (Int l = 0; l < BLK; l++)
            {
                Recon_R += DFT_Real[k * BLK + l] * cos(2.0 * PI * (double)(i * k + j * l) / (double)BLK) + DFT_Imag[k * BLK + l] * sin(2.0 * PI * (double)(i * k + j * l) / (double)BLK);

            }

        }

        if (Recon_R < 0)
            Recon_R = (int)(Recon_R - 0.5);

        else
            Recon_R = (int)(Recon_R + 0.5);

        img->Rec_img[(Blk_Row * BLK + i) * img->info.Ori_stride + (Blk_COl * BLK + j)] = CLIP(Recon_R);
    }
```

- Magnitude, Phase에서 불러온 Mag_buf, Pha_buf을 통해 IDFT를 수행하기 위한 DFT_Real, DFT_imag 구하기
- 이후 실수 부분(Mag_buf, cos), 허수 부분(Pha_buf, sin) 끼리의 연산을 통해 Recon_R에 누적합
- 이후 Clipping 과정을 통한 후 Rec_img에 저장

DCT_process

```
//DCT
Padding_buf = (UChar*)calloc(BLK * BLK, sizeof(UChar));
for (Int Blk_Row = 0; Blk_Row < img->info.height / BLK; Blk_Row++)
{
    for (Int Blk_COl = 0; Blk_COl < img->info.width / BLK; Blk_COl++)
    {
        memset(Padding_buf, 0, sizeof(UChar) * BLK * BLK);

        Focus_TMP = img->Ori_img + ((Blk_Row * BLK) * img->info.Ori_stride + (Blk_COl * BLK));          //왼쪽위
        for (int k = 0; k < BLK; k++)
            for (int l = 0; l < BLK; l++)
            {
                TMP = Focus_TMP + (k * img->info.Ori_stride + l);
                Padding_buf[k * BLK + l] = TMP[0];
            }

        DCT_Func(Padding_buf, BLK, Blk_Row, Blk_COl, img);
    }
}
free(Padding_buf);

//IDCT

DCT_buf = (Double*)calloc(BLK * BLK, sizeof(Double));
for (Int Blk_Row = 0; Blk_Row < img->info.height / BLK; Blk_Row++)
{
    for (Int Blk_COl = 0; Blk_COl < img->info.width / BLK; Blk_COl++)
    {
        memset(DCT_buf, 0, sizeof(Double) * BLK * BLK);

        DCT_Focus_TMP = img->DCT_img + ((Blk_Row * BLK) * img->info.Ori_stride + (Blk_COl * BLK));          //왼쪽위
        for (int k = 0; k < BLK; k++)
            for (int l = 0; l < BLK; l++)
            {
                DCT_TMP = DCT_Focus_TMP + (k * img->info.Ori_stride + l);
                DCT_buf[k * BLK + l] = DCT_TMP[0];
            }

        IDCT_Func(DCT_buf, BLK, Blk_Row, Blk_COl, img);
    }
```

- Padding_buf : BLK * BLK 크기의 메모리 할당
- memset : Padding_buf 값 초기화
- Focus_TMP : Ori_img에서 BLK * BLK 크기 배열의 첫번째 픽셀 주소
- 이후 TMP를 통해 Padding_buf에 픽셀 값 할당
- DCT_Func 진행

- memset : Mag_buf, Pha_buf 값 초기화
- DCT_Focus_TMP : DCT_img에서 BLK * BLK 크기 배열의 첫번째 픽셀 주소
- 이후 TMP를 통해 Mag_buf, Pha_buf 에 픽셀 값 할당
- IDCT_Func 진행

※IDCT 과정에서 DCT 후 값들이 Uchar형으로 변환되어 소수점 아래 값들이 손실 되지 않도록 Double형 이용

DCT_Func

```
for (v = 0; v < BLK; v++) {
    if (v == 0) {
        c_v = sqrt(0.5);
    }
    else {
        c_v = 1.0;
    }
    for (u = 0; u < BLK; u++) {
        if (u == 0) {
            c_u = sqrt(0.5);
        }
        else {
            c_u = 1.0;
        }
        temp = 0;

        for (y = 0; y < BLK; y++) {
            for (x = 0; x < BLK; x++) {
                temp += (double)buf[BLK * y + x] * cos((double)(2.0 * x + 1.0) * (double)u * PI / (double)(2.0 * BLK)) * cos((double)(2.0 * y + 1.0) * (double)v * PI / (double)(2.0 * BLK));
            }
        }
        DCT[BLK * v + u] = temp * c_v * c_u * 2.0 / sqrt(BLK * BLK);
    }
}

for (int i = 0; i < BLK; i++)
{
    for (int j = 0; j < BLK; j++)
    {
        img->DCT_img[(Blk_Row * BLK + i) * img->info.Ori_stride + (Blk_Col * BLK + j)] = DCT[BLK * i + j];
    }
}
```

- Pdf에 있는 수식을 이용해 DCT 진행
- DCT 수행 후 DCT_img에 저장

IDCT_Func

```
for (y = 0; y < BLK; y++) {
    for (x = 0; x < BLK; x++) {
        temp = 0;
        for (v = 0; v < BLK; v++) {
            if (v == 0) {
                c_v = sqrt(0.5);
            }
            else {
                c_v = 1.0;
            }
            for (u = 0; u < BLK; u++) {
                if (u == 0) {
                    c_u = sqrt(0.5);
                }
                else {
                    c_u = 1.0;
                }
                temp += (2.0 * c_v * c_u / sqrt(BLK * BLK) * buf[BLK * v + u] * cos((double)(2.0 * x + 1.0) * (double)u * PI / (double)(2.0 * BLK)) * cos((double)(2.0 * y + 1.0) * (double)v * PI / (double)(2.0
            }
        }

        if (temp < 0)
            temp = (int)(temp - 0.5);

        else
            temp = (int)(temp + 0.5);

        img->Rec_DCT_img[(Blk_Row * BLK + y) * img->info.Ori_stride + (Blk_COl * BLK + x)] = CLIP(temp);
```

- Pdf에 있는 수식을 이용해 IDCT 과정 진행
- 이후 Clipping을 거친 후 Rec_DCT_img에 저장

결과

DCT결과입니다 →

```
DFT 결과
MSE : 0.000000
PSNR : inf

영상 일치


DFT 결과
MSE : 0.000000
PSNR : inf

영상 일치

DFT 소요된 시간 : 3.931
start = 0
end   = 3931
```

```
DFT 결과
MSE : 0.000000
PSNR : inf

영상 일치


DFT 결과
MSE : 0.000000
PSNR : inf

영상 일치

DFT 소요된 시간 : 7652.905
start = 0
end   = 7652905
```

	Blk_SIZE=8	Blk_SIZE=512
DFT		
DCT		

- DFT, DCT 모두 육안으로 보기에 원본과 같은 영상이 나오고 실제 PSNR = inf, MSE = 0으로 측정
- 8x8 블록 단위 진행 후 소요된 시간 측정해보니 7.582s
- 512x512 블록 단위 진행 후 소요된 시간 측정해보니 7652.905s
- 큰 블록 단위로 변환 진행 시 복잡도가 증가함을 확인