# 数字图像处理代码

## 1.二维快速傅里叶变换（FFT2D）

```
void FFT2D(myComplex *src, myComplex *dst, int nr, int nc){
    myComplex* temp = (myComplex*)malloc(sizeof(myComplex)* nr*nc);
    for (int i = 0; i < nr*nc; i++){
        temp[i].real = src[i].real;
        temp[i].imagin = src[i].imagin;
    }
    // 先对每行进行一维FFT
    for (int i = 0; i < nr; i++)
        realFFT(&src[i*nc], &temp[i*nc], nc);

    //cout << "------------------------mid:-------------------------" << endl;
    //for (int i = 0; i < NUM; i++, cout << endl)
    //  Show_Complex(&temp[i*NUM], NUM);
    //cout << "------------------------mid:-------------------------" << endl;

    // 然后转置
    for (int i = 0; i < nr; i++){
        for (int j = i + 1; j < nc; j++){
            Swap_Complex(&temp[j*nr + i], &temp[i*nc + j]);
        }
    }
    // 再次对每行进行一维FFT，等同于对转置前的矩阵每列进行FFT
    for (int i = 0; i < nr; i++)
        realFFT(&temp[i*nc], &dst[i*nc], nc);

    free(temp);

    // 转置回来
    for (int i = 0; i < nr; i++){
        for (int j = i + 1; j < nc; j++){
            Swap_Complex(&dst[j*nr + i], &dst[i*nc + j]);
        }
    }
}
```

## 2.二维快速傅里叶反变换（IFFT2D）

```
void IFFT2D(myComplex *src, myComplex *dst, int N){
    myComplex* temp = (myComplex*)malloc(sizeof(myComplex)* N*N);
    // 先转置
    for (int i = 0; i < N; i++){
        for (int j = i + 1; j < N; j++){
            Swap_Complex(&src[j*N + i], &src[i*N + j]);
        }
    }
    // 再次对每行进行一维IFFT，等同于对转置前的矩阵每列进行IFFT
    for (int i = 0; i < N; i++)
        IFFT(&src[i*N], &temp[i*N], N);
```

```
    // 取矩阵的共轭
    //for (int i = 0; i < N*N; i++){
    //  getConjugate(&temp[i]);
    //}
    // 再次转置
    for (int i = 0; i < N; i++){
        for (int j = i + 1; j < N; j++){
            Swap_Complex(&temp[j*N + i], &temp[i*N + j]);
        }
    }
    // 再次对每行进行一维IFFT，等同于对原矩阵每行进行IFFT
    for (int i = 0; i < N; i++){
        IFFT(&temp[i*N], &dst[i*N], N);
    }
}
void FFT_Shift(myComplex *src, int nr, int nc){
    for (int i = 0; i < nr; i++){
        for (int j = 0; j < nc; j++){
            if ((i + j) % 2){
                src[i*nc + j].real = -src[i*nc +j].real;
            }
        }
    }
}
```

## 3.一维离散余弦变换（DCT1D）

```
void DCT(double *src, double *dst, int N){
    double dTemp;
    myComplex *temp = (myComplex*)malloc(sizeof(myComplex)*N*2);
    myComplex *temp2 = (myComplex*)malloc(sizeof(myComplex)*N * 2);
    memset(temp, 0, sizeof(myComplex)*N *2);
    memset(temp2, 0, sizeof(myComplex)*N*2);
    for (int i = 0; i < N; i++){
        temp[i].real = src[i];
        temp[N + i].real = 0;
    }

    dTemp = 1.0 / sqrt(N);

    for (int i = 0; i < N; i++)
        dst[0] += temp[i].real;
    dst[0] *= dTemp;

    realFFT(temp, temp2, N<<1);
    dTemp *= sqrt(2);

    myComplex *t = new myComplex();
    for (int i = 1; i < N; i++){
        getWn(i, N<<2, t);
        Mul_Complex(t, &temp2[i], t);
        dst[i] = dTemp * (t->real);
    }

    free(temp);
    free(temp2);
}
```

## 4.一维离散余弦反变换（IDCT1D）

```
void IDCT(double *src, double *dst, int N){
    double N1 = 1 / sqrt(N);
    double N2 = sqrt(2) / sqrt(N);
    myComplex * t = new myComplex();
    myComplex * tC = new myComplex();
    myComplex *temp = (myComplex*)malloc(sizeof(myComplex)*N * 2);
    myComplex *temp2 = (myComplex*)malloc(sizeof(myComplex)*N * 2);
    double *tsrc = (double*)malloc(sizeof(double)*N * 2);
    memset(tsrc, 0, sizeof(double)*N * 2);
    for (int i = 0; i < N; i++)
        tsrc[i] = src[i];
    for (int i = 0; i < (N<<1); i++){
        getWn(-i, N << 2, t); // 这里要用-i

        tC->real = tsrc[i];
        tC->imagin = 0;

        Mul_Complex(t, tC, &temp[i]);
    }
    IFFT(temp, temp2, N << 1);

    for (int i = 0; i < N; i++){
        dst[i] = (N1 - N2) * tsrc[0] + N2 * temp2[i].real * (N<<1);
    }

    free(temp);
    free(temp2);
    free(tsrc);
}
```

## 5.二维离散余弦变换（DCT2D）

```
void DCT2D(double *src , double *dst, int nr, int nc){
    double *temp = (double *)malloc(sizeof(double)*nr*nc);
    for (int i = 0; i < nr; i++){
        DCT(&src[i*nc], &temp[i*nc], nc);
    }
    // 然后转置
    for (int i = 0; i < nr; i++){
        for (int j = i + 1; j < nc; j++){
            double t = temp[j*nr + i];
            temp[j*nr + i] = temp[i *nc + j];
            temp[i*nc + j] = t;
            //Swap_Complex(&temp[j*nr + i], &temp[i*nc + j]);
        }
    }
    for (int i = 0; i < nc; i++){
        DCT(&temp[i*nr], &dst[i*nr], nr);
    }
    for (int i = 0; i < nc; i++){
        for (int j = i + 1; j < nr; j++){
            double t = dst[i*nr + j];
```

```
            dst[i*nr + j] = dst[j *nr + i];
            dst[j*nr + i] = t;
        }
    }
    free(temp);
}
```

## 6.二维离散余弦反变换（IDCT2D）

```
void IDCT2D(double *src, double*dst, int nr, int nc){
    double *temp = (double *)malloc(sizeof(double)*nr*nc);
    for (int i = 0; i < nc; i++){
        for (int j = i + 1; j < nr; j++){
            double t = src[i*nr + j];
            src[i*nr + j] = src[j *nr + i];
            src[j*nr + i] = t;
        }
    }
    for (int i = 0; i < nr; i++){
        IDCT(&src[i*nc], &temp[i*nc], nc);
    }
    for (int i = 0; i < nr; i++){
        for (int j = i + 1; j < nc; j++){
            double t = temp[j*nr + i];
            temp[j*nr + i] = temp[i *nc + j];
            temp[i*nc + j] = t;
            //Swap_Complex(&temp[j*nr + i], &temp[i*nc + j]);
        }
    }
    for (int i = 0; i < nc; i++)
        IDCT(&temp[i*nr], &dst[i*nr], nr);
}
```

## 7.一维快速傅里叶变换（FFT1D）

网络版

```
void FFT(myComplex* src, int N, int I, int r, int idx, int MAX_N){
    if (N == 1)return;
    myComplex *tt1 = new myComplex();
    myComplex *tt2 = new myComplex();
    myComplex* t1 = new myComplex();
    //cout << "idx=" << idx << endl;
    for (int i = 0; i < (N >> 1); i ++){
        int k = i + idx;
        int dst = k + (MAX_N >> I);
        int P = getP(k, I, r);
        int dstP = getP(dst, I, r);

        //伪代码temp[k] = src[k] + getWn(P) * src[dst];
        getWn(P, MAX_N, t1);
        Mul_Complex(t1, &src[dst], t1);
        Add_Complex(&src[k], t1, tt1);
        //伪代码temp[dst] = src[k] - getWn(dstP) * src[dst];
        getWn(dstP, MAX_N, t1);
        Mul_Complex(t1, &src[dst], t1);
```

```
        //Sub_Complex(&src[k], t1, &temp[dst]); //  不需要用到减法，因为W4 = -W0
        Add_Complex(&src[k], t1, tt2);

        Copy_Complex(tt1, &src[k]);
        Copy_Complex(tt2, &src[dst]);
    }
    //free(temp);
    delete(tt1);
    delete(tt2);
    delete(t1);
    FFT(src, N>>1, I+1, r, idx, MAX_N);
    FFT(src, N>>1, I+1, r, idx + (N>>1), MAX_N);
}

void realFFT(myComplex *src, myComplex *dst, int N){
    int r = log10(N) / log10(2); // log2(N) = lg(N) / lg(2)

    //Show_Complex(temp, N);

    FFT(src,N,1,r,0, N);
    for (int i = 0; i < N; i++){
        //cout << reverseNum(i, r) << ' ';
        Copy_Complex(&src[i], &dst[reverseNum(i, r)]);
    }
}
```

老师的版本

```
//**************************************************************************
//**    下面是一维n点FFT变换Fft1D()，其中*fr和*fi分别为输入信号的实部和虚部，   **
//**    同时也是对应的输出频域信号的实部和虚部。在调用之前，需先计算复常数数 **
//**    组W[n/2]的值：                                                    **
//**    complex<double> *W;                                               **
//**    W=new complex<double>[n/2];                                        **
//**    double t=2.0*PI/n;                                                 **
//**    for(int i=0;i<n/2;i++)                                             **
//**    W[i]=complex<double> (cos(t*i),-sin(t*i));                         **
//**……                                                    **
//**delete W;                                                    **

//**************************************************************************

void FastDCT::FFT1D(double *fr, double *fi, LONG n, complex<double> *W)
{
int i,j,k,l,le,le1,ip,ic;
double t1,t2;
int m=(int)(log10(n)/log10(2));
int n2=n/2;

// 重排序
j=0;
for(i=1;i<n-1;i++)
{
    k=n2;
    while(k<=j){j-=k;k/=2;}
    j+=k;
```

```
    if(i<j)
    {
        t1=*(fr+j);
        *(fr+j)=*(fr+i);
        *(fr+i)=t1;
        t2=*(fi+j);
        *(fi+j)=*(fi+i);
        *(fi+i)=t2;
    }
}
l=1;
for(i=0;i<m;i++)
{
    le=l;
    l*=2;  //l=2**l
    for(j=0;j<le;j++)
    {
        le1=n/l;
        ic=j*le1;
        for(k=j;k<n;k+=l)
        {
            ip=k+le;
            t1=(*(fr+ip))*(w[ic].real())-(*(fi+ip))*(w[ic].imag());
            t2=(*(fr+ip))*(w[ic].imag())+(*(fi+ip))*(w[ic].real());
            *(fr+ip)=*(fr+k)-t1;
            *(fi+ip)=*(fi+k)-t2;
            *(fr+k)+=t1;
            *(fi+k)+=t2;
        }
    }
}

/*******************
for(i=0;i<n;i++)
{
    *(fr+i)/=n;
    *(fi+i)/=n;
}
*******************/

return;

}
```

# 8.一维快速傅里叶反变换（IFFT1D）

网络版

```
void IFFT(myComplex* src, myComplex *dst, int N){
    double t = 1.0 / N;
    for (int i = 0; i < N; i++){
        getConjugate(&src[i]);
    }
    realFFT(src, dst, N);
    for (int i = 0; i < N; i++){
        getConjugate(&dst[i]);
        dst[i].real *= t;
        dst[i].imagin *= t;
    }
}
```

老师的版本

```
BOOL FastDCT::IFFT1D(double *fr, double *fi, LONG n, complex<double> *W)
{
    for(int i=0;i<n;i++) *(fi+i)=-*(fi+i);
    FFT1D(fr,fi,n,W);

    for(i=0;i<n;i++)
    {
        *(fr+i)/=n;
        *(fi+i)/=n;
        *(fi+i)=-*(fi+i);
    }

    return TRUE;
}
```