Digital Image Processing

Assignment 2 - VQ, DCT, and Wavelet

1. Code

- Language: C/C++
- Libraries:
  - <windows.h>: for image file loading and saving support
  - <stdio.h>, <iostream>: input, output, file loading and saving
  - <math.h>, <string> <time.h>: computation and function support
- Implementation:
  - FDCT, IDCT
  - fast FDCT, fast IDCT
  - Wavelet transformation
  - Execution time comparison

DIP_assignment2.cpp

```
#include <windows.h>
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <string>
#include <time.h>
using namespace std;

//constant
const int N = 128;
const double PI = 3.14159265359;

//choose file and algorithm
const string file = "2";
const int _DCT = 0, _FASTDCT = 1, _DWT = 2;
const int ALGO = 2;

//variables for DCT
double ods2N;
double cosList[N][N];
double cList[N];

//variables for fast DCT
double C[N][N];
double Ct[N][N];

//variables for image loading and saving
BITMAPFILEHEADER bmf;
BITMAPINFOHEADER bmi;
RGBQUAD pal[256];

//transform 2D location to 1D location
inline int loc(int i, int j)
{
    return i * N + j;
}

//Load a image into an integer array
int loadImg(char* filename, int width, int height, int imgData[])
{
    FILE* fp;
    errno_t err;
    unsigned char* data;
    int i, j, count;
```

```
    err = fopen_s(&fp, filename, "rb");
    if (err != 0)
        return 0;

    data = (unsigned char*)malloc(width);

    count = fread(&bmf, 1, sizeof(bmf), fp);
    count = fread(&bmi, 1, sizeof(bmi), fp);
    count = fread(&pal, 1, sizeof(pal), fp);

    count = 0;
    for (j = 0; j < height; j++)
    {
        int len = fread(data, 1, width, fp);
        for (i = 0; i < len; i++)
            imgData[(height - 1 - j) * width + i] = (int)(unsigned int)data[i];
        count += len;
    }

    free(data);
    fclose(fp);
    return count;
}

//Saving a integer array into a .bpm image file
int saveImg(char* filename, int width, int height, int imgData[])
{
    FILE* fp;
    errno_t err;
    unsigned char* data;
    int i, j, count;

    err = fopen_s(&fp, filename, "wb");
    if (err != 0)
        return 0;

    data = (unsigned char*)malloc(width);

    count = fwrite(&bmf, 1, sizeof(bmf), fp);
    count = fwrite(&bmi, 1, sizeof(bmi), fp);
    count = fwrite(&pal, 1, sizeof(pal), fp);

    count = 0;
    for (j = 0; j < height; j++)
    {
        for (i = 0; i < width; i++)
        {
            int t = imgData[(height - 1 - j) * width + i];
            if (t < 0)
                t = 0;
            else if (t > 255)
                t = 255;
            data[i] = t;
        }

        int len = fwrite(data, 1, width, fp);
        count += width;
    }

    free(data);
    fclose(fp);
    return count;
}

//Initialize some variables for DCT algorithms
void init()
{
    //variables for DCT
    ods2N = 2.0 / N;
    int i, j;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
            cosList[i][j] = cos((2 * i + 1) * j * PI / (2.0 * N));
    }

    cList[0] = 1 / sqrt(2);
    for (i = 1; i < N; i++)
        cList[i] = 1;

    //variables for fast DCT
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            if (i == 0)
            {
```

```
                C[i][j] = 1 / sqrt(N);
                Ct[j][i] = C[i][j];
            }
            else
            {
                C[i][j] = sqrt(2.0 / N) * cos((2 * j + 1) * i * PI / (2 * N));
                Ct[j][i] = C[i][j];
            }
        }
    }
}

//Implemetation of FDCT
//Input a bit array of image, and output a spectrum array
void FDCT(int input[], int output[])
{
    int i, j;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            double temp = 0;
            int x, y;
            for (x = 0; x < N; x++)
            {
                for (y = 0; y < N; y++)
                    temp += cosList[x][i] * cosList[y][j] * (input[x * N + y] - 128);
            }

            temp *= ods2N * cList[i] * cList[j];
            output[i * N + j] = round(temp);
        }
    }
}

//Implemetation of IDCT
//Input a spectrum array, and output a bit array of image
void IDCT(int input[], int output[])
{
    int x, y;
    for (x = 0; x < N; x++)
    {
        for (y = 0; y < N; y++)
        {
            double temp = 0;

            int i, j;
            for (i = 0; i < N; i++)
            {
                for (j = 0; j < N; j++)
                {temp += cList[i] * cList[j] *
                        cosList[x][i] * cosList[y][j] *
                        input[i * N + j];
                }
            }

            temp *= ods2N;
            temp += 128;

            //adjust the bits that exceed 0~255
            if (temp < 0)
                temp = 0;
            else if (temp > 255)
                temp = 255;
            output[x * N + y] = round(temp);
        }
    }
}

//Re-implementation of FDCT, but with fast algorithm
void fastFDCT(int input[], int output[])
{
    double temp[N][N];
    int i, j, k;

    //implement input * Ct
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            temp[i][j] = 0;
            for (k = 0; k < N; k++)
                temp[i][j] += (input[i * N + k] - 128) * Ct[k][j];
        }
    }

    //implement C * (input * Ct)
```

```
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            double temp2 = 0;
            for (k = 0; k < N; k++)
                temp2 += C[i][k] * temp[k][j];
            output[i * N + j] = round(temp2);
        }
    }
}

//Re-implementation of FDCT, but with fast algorithm
void fastIDCT(int input[], int output[])
{
    double temp[N][N];
    int i, j, k;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            temp[i][j] = 0;
            for (k = 0; k < N; k++)
                temp[i][j] += input[i * N + k] * C[k][j];
        }
    }

    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            double temp2 = 0;
            for (k = 0; k < N; k++)
                temp2 += Ct[i][k] * temp[k][j];
            temp2 += 128;

            if (temp2 < 0)
                temp2 = 0;
            else if (temp2 > 255)
                temp2 = 255;
            output[i * N + j] = round(temp2);
        }
    }
}

//Implemetation of WT
//Input a bit array of image, and output a spectrum array
void DWT(int input[], int output[])
{
    int* temp = (int*)malloc(N * N * sizeof(int));

    int i, j;

    //horizontal transformation
    for (i = 0; i < N; i++)
    {
        int k = 0;
        for (j = 0; j < N; j+=2)
        {
            int a = input[loc(i, j)];
            int b = input[loc(i, j + 1)];
            temp[loc(i, k)] = (a + b);
            temp[loc(i, k + N / 2)] = (a - b);
            k++;
        }
    }

    //vertical transformation
    for (j = 0; j < N; j++)
    {
        int k = 0;
        for (i = 0; i < N; i += 2)
        {
            int a = temp[loc(i, j)];
            int b = temp[loc(i + 1, j)];
            output[loc(k, j)] = (a + b);
            output[loc(k + N / 2, j)] = (a - b);
            k++;
        }
    }
}

//Implemetation of IWT
//Input a spectrum array, and output a bit array of image
void IDWT(int input[], int output[])
{
    int* temp = (int*)malloc(N * N * sizeof(int));
```

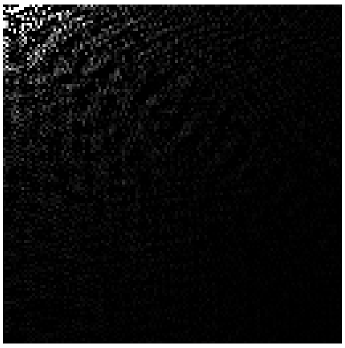
```
    int i, j;

    //vertical transformation
    for (j = 0; j < N; j++)
    {
        int k = 0;
        for (i = 0; i < N / 2; i++)
        {
            int sum = input[loc(i, j)];
            int diff = input[loc(i + N / 2, j)];
            temp[loc(k, j)] = (sum + diff)/2;
            temp[loc(k + 1, j)] = (sum - diff) /2;
            k+=2;
        }
    }

    //horizontal transformation
    for (i = 0; i < N; i++)
    {
        int k = 0;
        for (j = 0; j < N / 2; j++)
        {
            int sum = temp[loc(i, j)];
            int diff = temp[loc(i, j + N / 2)];
            output[loc(i, k)] = (sum + diff) /2;
            output[loc(i, k + 1)] = (sum - diff)/2;
            k+=2;
        }
    }
}

int main(int argc, char* argv[])
{
    //allocate memory
    int* srcImage = (int*)malloc(N * N * sizeof(int));
    int* specImage = (int*)malloc(N * N * sizeof(int));
    int* dstImage = (int*)malloc(N * N * sizeof(int));

    //set time flag
    double startTime, endTime;

    //load image
    cout << "Loading image..." << endl;
    int count = loadImg((char*)("img/" + file + ".bmp").c_str(), N, N, srcImage);
    if (count == 0)
    {
        cout << "Fail to load image" << endl;
        exit(1);
    }

    //do algorithm
    switch (ALGO)
    {
    case _DCT:
        cout << "Initialzing..." << endl;
        init();

        startTime = clock();
        cout << "Doing FDCT..." << endl;
        FDCT(srcImage, specImage);
        cout << "Doing IDCT..." << endl;
        IDCT(specImage, dstImage);
        endTime = clock();

        cout << "Saving output image..." << endl;
        count = saveImg((char*)("img/" + file + "_DCT_output.bmp").c_str(), N, N, dstImage);
        if (count == 0)
        {
            cout << "Fail to save image" << endl;
            exit(1);
        }

        cout << "Saving spectrum image..." << endl;
        count = saveImg((char*)("img/" + file + "_DCT_spec.bmp").c_str(), N, N, specImage);
        if (count == 0)
        {
            cout << "Fail to save image" << endl;
            exit(1);
        }

        cout << "Done!!!" << endl;
        cout << "Use " << endTime - startTime << " ms" << endl;
        break;
    case _FASTDCT:
        cout << "Initialzing..." << endl;
        init();
```

```
        startTime = clock();
        cout << "Doing fast FDCT..." << endl;
        fastFDCT(srcImage, specImage);
        cout << "Doing fast IDCT..." << endl;
        fastIDCT(specImage, dstImage);
        endTime = clock();

        cout << "Saving output image..." << endl;
        count = saveImg((char*)("img/" + file + "_FASTDCT_output.bmp").c_str(), N, N,
dstImage);
        if (count == 0)
        {
            cout << "Fail to save image" << endl;
            exit(1);
        }

        cout << "Saving spectrum image..." << endl;
        count = saveImg((char*)("img/" + file + "_FASTDCT_spec.bmp").c_str(), N, N,
specImage);
        if (count == 0)
        {
            cout << "Fail to save image" << endl;
            exit(1);
        }

        cout << "Done!!!" << endl;
        cout << "Use " << endTime - startTime << " ms" << endl;
        break;
    case _DWT:
        cout << "Doing DWT..." << endl;
        DWT(srcImage, specImage);

        cout << "Saving spectrum image..." << endl;
        count = saveImg((char*)("img/" + file + "_DWT_output.bmp").c_str(), N, N,
specImage);
        if (count == 0)
        {
            cout << "Fail to save image" << endl;
            exit(1);
        }

        cout << "Doing IDWT..." << endl;
        IDWT(specImage, dstImage);

        cout << "Saving output image..." << endl;
        count = saveImg((char*)("img/" + file + "_IDWT_output.bmp").c_str(), N, N,
dstImage);
        if (count == 0)
        {
            cout << "Fail to save image" << endl;
            exit(1);
        }

        cout << "Done!!!" << endl;
    }

    return 0;
}
```

2. Result

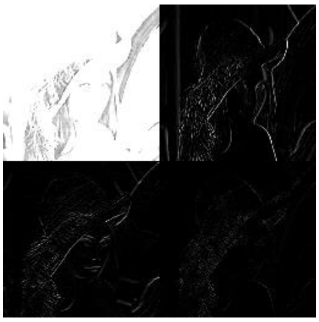
DCT & IDCT

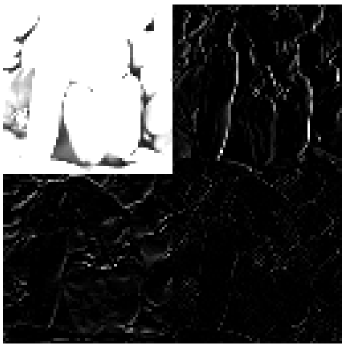
Source image	After DCT	After IDCT
		

Source image	After DCT	After IDCT
		

Source image	After DCT	After IDCT
		

WT & IWT

Source image	After WT	After inverse WT
		

Source image	After WT	After inverse WT
		

Source image	After WT	After inverse WT
		

Execution time comparison

Execution time (ms)	DCT	Fast DCT
64x64	275	7
128x128	2344	48
256x256	40867	523

3. Findings & Conclusion

- DCT transformation has a great feature of **energy concentration**. Frequency information will be concentrated on the low-frequency part (top-left corner of the spectrum).
- A very famous application of DCT is **JPEG**, which uses DCT to do the lossy compression. The information of low-frequency part dominates the vision of human eyes, so if we delete the high-frequency information, the loss will be very small, but can reduce the data size significantly.
- According to the experiment result, these two algorithms have significant difference in the speed--the fast DCT is **40~70 times faster** than the normal DCT. In normal DCT, each point should take cosine value of each location, so the complexity is O(n^4). However, the fast DCT only does matrix multiplication for twice, which is O(n^3). So, we can predict that fast DCT will be much faster than the normal DCT. When we want to deal with an image in large size, we had better to use fast DCT.
- The WT also has a great feature of energy concentration. I think the biggest advantage of WT is speed. **Its complexity is O(n^2)**, even better than fast DCT. Moreover, it basically only needs +/- to complete the calculation without complex math, which can reduce execution time a lot.