西北工业大学计算机学院

《多媒体技术》实验报告

学　　号：　　2019302863

姓　　名：　　李奇

实验时间：　　2021.11.17

实验地点：　　实验大楼 209-3

实验题目：　　直方图均衡增强 & 2D DCT 变换

西北工业大学

2021 年 11 月

一、 实验目的及要求。

1. 了解直方图的定义。

2. 了解累积分布函数的相关知识。

3. 熟练掌握 2D DCT 变换的原理和流程。

二、 实验内容与操作步骤。

1. 使用直方图均衡增强测试图片 testimg.txt。

2. 使用 2D DCT 变换将 testimg.txt 转换到频域，并输出结果。要求以 8x8 为处理单元。

三、 实验结果分析。

1. 直方图的定义：

直方图(Histogram)，又称质量分布图，是一种统计报告图，由一系列高度不等的纵向条纹或线段表示数据分布的情况。 一般用横轴表示数据类型，纵轴表示分布情况。图像直方图由于其计算代价较小，且具有图像平移、旋转、缩放不变性等众多优点，广泛地应用于图像处理的各个领域，特别是灰度图像的阈值分割、基于颜色的图像检索以及图像分类。

2. 使用直方图均衡增强测试图片 testimg.txt。

实现图像增强有空间域法和频率域法两大类方法，本题所采用的直方图均衡增强方法属于空间域法的一种。在均衡化过程中，必须要保证两个条件：

一是像素无论怎么映射，一定要保证原来的大小关系不变，较亮的区域，依旧是较亮的，较暗依旧暗，只是对比度增大，绝对不能明暗颠倒；二是如果是八位图像，那么像素映射函数的值域应在 0 和 255 之间的，不能越界。

综合以上两个条件，累积分布函数是个好的选择，因为累积分布函数是单调增函数（控制大小关系），并且值域是 0 到 1（控制越界问题），所以直方图均衡化中使用的是累积分布函数。

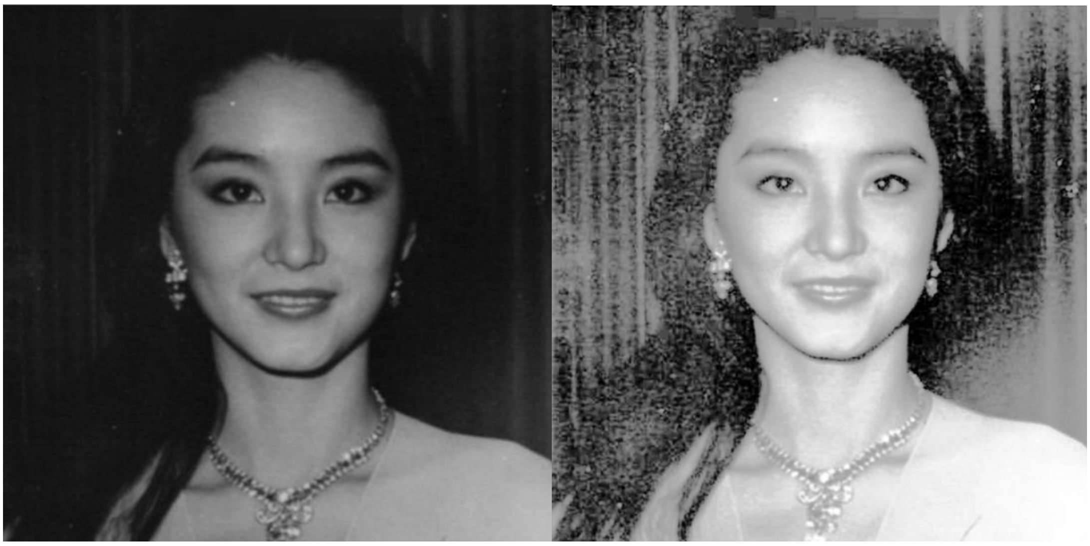
本题在实现时用 c 语言将 testimg.txt 进行直方图增强，代码主要分为以下几个部分：读取数据，统计不同灰度出现个数，计算不同灰度出现概率，用数组存储概率直方图，将原来灰度进行变化，写入新数据，生成名为 data.txt 的文件。具体代码如下：

```
#define _CRT_SECURE_NO_WARNINGS
```

```
#define _CRT_NONSTDC_NO_DEPRECATE
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
using namespace std;
int PIXEL[256];
int NUMPIXEL=0;
double HISTOGRAM[256];
double SUM_HISTOGRAM[256];
int  CHANGE_PIXEL[256];
void refreshPIXEL()
{
    for (int i=0; i <=255; i++)
    {
        PIXEL[i] = 0;
    }
}
void countNUMPIXEL()
{
    for (int i=0; i <=255; i++)
    {
        if (PIXEL[i] != 0)
        {
            NUMPIXEL=NUMPIXEL+PIXEL[i];
        }
    }
}
int main()
{
    FILE* fp1 = NULL;
    FILE* fp2 = NULL;
    int middle = 0;
    refreshPIXEL();
    freopen("C:\\Users\\61060\\Desktop\\NPU-LQ\\大三上\\多媒体
\\LiQi_Image\\work1\\testimg.txt", "r" , stdin);
    while (1)
    {
```

```
        cin >> middle;
        if (middle >= 0)
        {
            PIXEL[middle]++;
        }
        else break;
    }
    countNUMPIXEL();
    int i=0; int j=0;
    for (i = 0; i <= 255; i++)
    {
        HISTOGRAM[i] =( PIXEL[i]*1.000) / (NUMPIXEL*1.0000);
    }
    for (i = 1; i <= 255; i++)
    {
        SUM_HISTOGRAM[i] = HISTOGRAM[i] + SUM_HISTOGRAM[i - 1];
    }
    for (i = 1; i <= 255; i++)
    {
        CHANGE_PIXEL[i] = SUM_HISTOGRAM[i] * 255.0000;
    }
    middle = 0;
    freopen("C:\\Users\\61060\\Desktop\\NPU-LQ\\大三上\\多媒体
\\LiQi_Image\\work1\\testimg.txt", "r" , stdin);
    freopen("C:\\Users\\61060\\Desktop\\NPU-LQ\\大三上\\多媒体
\\LiQi_Image\\work1\\data.txt", "w", stdout);
    int change_len=1;
    for (i = 1; i <= 512; i++)
    {
        for (j = 1; j <= 512; j++)
        {
            cin >> middle;
            cout << CHANGE_PIXEL[middle]<<"\t";
        }
        cout << endl;
   }
    return 0;
}
```

将生成的数据可视化后生成图像，该图像和原始图像进行对比如下图所示：



原始图像 vs 直方图均衡增强图像

3.使用 2D DCT 变换将 testimg.txt 转换到频域，并输出结果。要求以 8x8 为处理单元。

首先需要清楚二维离散余弦变换在一维下的情况，表达式如下：

$$F(0)=\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}f(x)$$

$$F(u)=\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\cos\frac{(2x+1)u\pi}{2N}$$

式中 F(u)是第 u 个余弦变换值，u 是广义频率变量，u=1,2,….,N-1;f(x)是时域 N 点序列。x= 1,2,….,N-1。

更为简洁的定义离散余弦变换是采用矩阵式定义。根据以上公式定义可知，我们可以来推导一下，DCT 变换可以用矩阵的形式表示出来，F(u)为变换域矩阵，是时域 f(x)与 A 矩阵计算的结果；A 为变换系数矩阵,当 N 取定值时，A 就是一个常量矩阵；f(x)为时域数据矩阵，即需要转换到变换域的原始数据，则一维离散余弦变换的矩阵定义式可写成下方表达式：

$$F=A[f(x)]$$

二维余弦变换可以写成如下形式：

$$F(0,0)=\frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}f(x,y)$$

$$F(0,v)=\frac{\sqrt{2}}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cdot\cos\frac{(2y+1)v\pi}{2N}$$

$$F(u,0)=\frac{\sqrt{2}}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\frac{(2x+1)u\pi}{2N}$$

$$F(u,v)=\frac{2}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\frac{(2x+1)u\pi}{2N}\cdot\cos\frac{(2y+1)v\pi}{2N}$$

最后一个公式是二维离散余弦变换的正变换公式，其中 f(x,y)是空间域一个 N*N 的二维向量元素，即一个 N*N 的矩阵，x,y = 0,1,2，…，N-1；F(U,V)是经计算后得到的变换域矩阵，u,v = 0,1,2，… N-1.求和可分性是二维离散余弦变换的一个重要特征，因此可用下式表示：

$$\mathrm{F(u,v)}=\frac{2}{N}\sum_{x=0}^{N-1}\cos\frac{(2x+1)u\pi}{2N}\left\{\sum_{y=0}^{N-1}\mathrm{f(x,y)}\cos\frac{(2x+1)v\pi}{2N}\right\}$$

由一维和二维的离散余弦变换公式性质可以推到得知，二维离散余弦变换也可写成矩阵相乘形式：

$$F=A[f(x,y)]A^T$$

其中 A 为一维离散余弦变换的变换系数矩阵，AT 是 A 的转置矩阵。

经过上述分析，可以写出相应的实现代码如下，该代码实现了对图像进行 DCT 变换，并生成了名为 data2.txt 的文件：

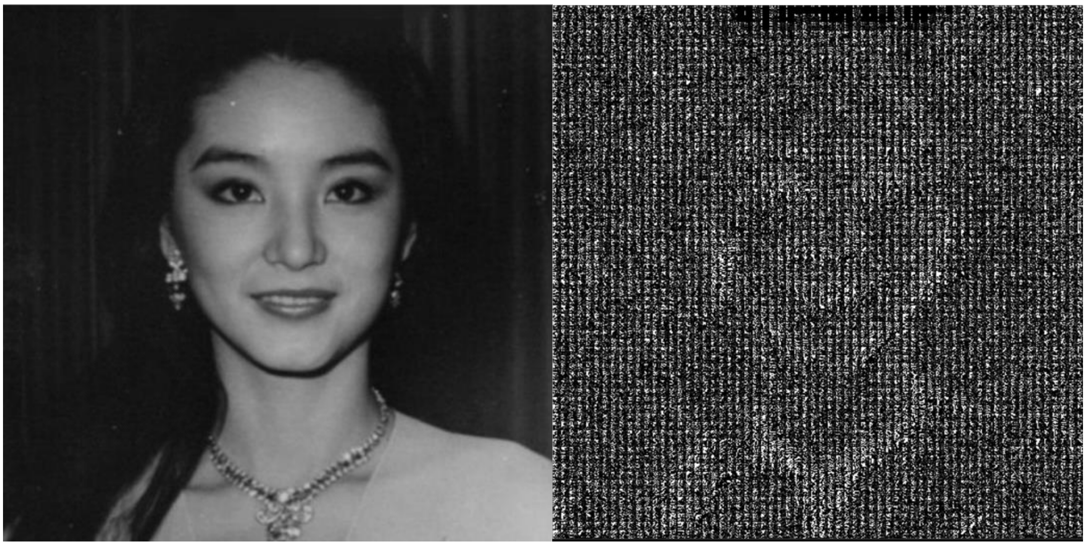
```
#define _CRT_SECURE_NO_WARNINGS
#define _CRT_NONSTDC_NO_DEPRECATE
#include <iostream>
#include <memory.h>
#include <math.h>
#include <time.h>
#include <iomanip>
using namespace std;
#define PI (3.1415926)
#define N (8)    /* 当前切成 8x8 的块进行 DCT 变换 */
int WIDTH = 512, HEIGHT = 512;
```

```
int MAT_SIZE = 512;
float DCT_Mat[512][512]; //定于变换矩阵
float DctMap[512][512];  //输入矩阵，计算结束后为输出矩阵
float DctMapTmp[512][512];  //矩阵运算时用的中间矩阵
/*
*读取像素
*/
void readFile()
{
    freopen("C:\\Users\\61060\\Desktop\\NPU-LQ\\大三上\\多媒体
\\LiQi_Image\\work1\\testimg.txt", "r", stdin);
    for (int i = 0; i < 512; i++)
    {
        for (int j = 0; j < 512; j++)
        {
            cin >> DctMap[i][j];
        }
    }
}
void TransMat()
{
    int i, j;
    float a;
    for (i = 0; i < MAT_SIZE; i++)
    {
        for (j = 0; j < MAT_SIZE; j++)
        {
            a = 0;
            if (i == 0)
            {
                a = sqrt((float)1 / MAT_SIZE);
            }
            else
            {
                a = sqrt((float)2 / MAT_SIZE);
            }
            DCT_Mat[i][j] = a * cos((j + 0.5) * PI * i / MAT_SIZE);
//变换矩阵
```

```
        }
    }
}
void DCT()
{
    float t = 0;
    int i, j, k;
    for (i = 0; i < MAT_SIZE; i++)  //相当于 A*I
    {
        for (j = 0; j < MAT_SIZE; j++)
        {
            t = 0;
            for (k = 0; k < MAT_SIZE; k++)
            {
                t += DCT_Mat[i][k] * DctMap[k][j]; //矩阵的乘法，
DCT_Mat 的第 i 行乘 DctMap 的第 j 列
            }
            DctMapTmp[i][j] = t;
        }
    }
    for (i = 0; i < MAT_SIZE; i++)  //相当于（A*I）后再*A‘
    {
        for (j = 0; j < MAT_SIZE; j++)
        {
            t = 0;
            for (k = 0; k < MAT_SIZE; k++)
            {
                t += DctMapTmp[i][k] * DCT_Mat[j][k];
            }
            DctMap[i][j] = t;
        }
    }
}
int main(int argc, char* argv[])
{
    readFile();
    TransMat();
    DCT();
```

```
    freopen("C:\\Users\\61060\\Desktop\\NPU-LQ\\大三上\\多媒体\\LiQi_Image\\work1\\data2.txt", "w", stdout);
    for (int i = 0; i < MAT_SIZE; i++)
    {
        for (int j = 0; j < MAT_SIZE; j++)
        {
            cout << DctMap[i][j] << "\t";
        }
        cout << endl;
    }
}
```

将生成的数据可视化后生成图像，该图像和原始图像进行对比如下图所示：

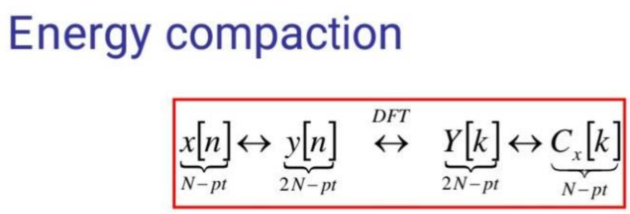


原始图像 vs DCT 图像

四、实验中存在问题及改进。

DCT 为什么具有更好的频域能量聚集度？

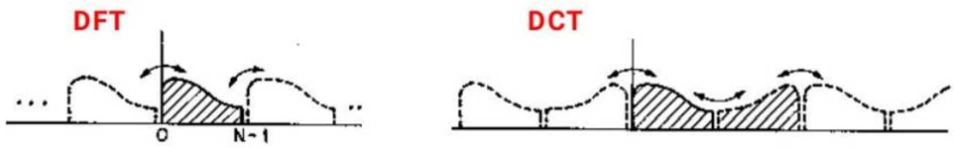
dct 是对信号进行先镜像之后再周期扩展，这样做的好处在于扩展的信号间实现了平滑的过度，而直接周期扩展会出现跳变，这种跳变在频域就对应着高频的分量。



- to understand the energy compaction property
  - we start by considering the sequence y[n] = x[n]+x[2N-1-n]
  - this just consists of adding a mirrored version of x[n] to itself

  - next we remember that the DFT is identical to the DFS of the periodic extension of the sequence
  - let's look at the periodic extensions for the two cases
    - when transform is DFT: we work with extension of x[n]
    - when transform is DCT: we work with extension of y[n]

- the two extensions are



  - note that in the DFT case the extension introduces discontinuities
  - this does not happen for the DCT, due to the symmetry of y[n]
  - the elimination of this artificial discontinuity, which contains a lot of high frequencies,
  - is the reason why the DCT is much more efficient