# Multimedia Technologies

# Project Report

## 1. Members

| 序号 | 学号 | 专业班级 | 姓名 | 性别 | 分工 |
|---|---|---|---|---|---|
| 1 | 3200102708 | 计科 2002 | 刘思锐 | 男 | 独立完成 |
| | | | | | |
| | | | | | |
| | | | | | |

## 2. Project Introduction

### 2.1 工作简介

本次实验的主题是实现与 JPEG 类似的有损图像压缩算法。实验使用 C++进行编程，利用了 OpenCV 库提供的图片源文件读取和矩阵运算等基础支持函数，但核心算法部分，例如颜色下采样、DCT 变换、霍夫曼编码等均为自己编写。

### 2.2 开发环境和运行说明

| 开发环境 | |
|---|---|
| 操作系统 | MacOS Ventura 13.3.1 |
| CPU | Apple Silicon M1 |
| 内存 | 16G |
| 开发工具 | Visual Studio Code |
| 编译器 | clang-1403.0.22.14.1 |

| 运行环境 | |
|---|---|
| 操作系统 | 任意（非 darwin 环境需重新编译） |
| CPU | 任意（非 arm 架构需重新编译） |
| 内存 | 4G |
| 环境依赖 | OpenCV 3.4.x |

工程附带上述开发环境下的 CMakeList 文件，可以使用如下命令重新编译：

```
$ mkdir build && cd build
$ cmake ../
$ make
```

编码器可执行文件名为 encode2708，使用方法如下，其中源文件可以是任意 OpenCV 支持打开的图片格式：

```
$ ./encode2708 [source picture path] [saved file name]
```

解码器可执行文件名为 decode2708，使用方法如下：

```
$ ./decode2708 [source file path]
```

项目自带的可执行文件使用动态连接，如无法运行请检查 OpenCV 环境配置是否正确。

# 3. Technical Details

## 3.1 理论基础

### 3.1.1 DCT 变换

一维的 DCT 变换公式如下：

$$F(u) = c(u) \sum_{i=0}^{N-1} f(i) cos[\frac{(i+0.5)u\pi}{N}]$$

$$c(u) = \begin{cases} \sqrt{\frac{1}{N}}, & u = 0 \\ \sqrt{\frac{2}{N}}, & u \neq 0 \end{cases}$$

其本质上在做信号分解，将一个原始信号分解为直流信号和若干正交的余弦信号的叠加，算法输出向量的含义是不同分量的系数。DCT 可以看作傅立叶变换的一种特殊情况，因此一种可行的实现方式是基于 FFT 稍作变形。

二维 DCT 则是在一维的基础上再进行一次 DCT 变换，两次变换方向相互正交。公式如下：

$$F(u,v) = c(u)c(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i) cos[\frac{(i+0.5)u\pi}{N}] cos[\frac{(j+0.5)v\pi}{N}]$$

基于二维 DCT 是两次方向相互正交的一维 DCT 叠加，引入矩阵 A：

$$A(i,j) = c(i)cos[\frac{(j+0.5)i\pi}{N}]$$

则二维 DCT 变换可以极其简便的表达为如下形式：

$$F = AfA^T$$

且观察到 A 是一个正交矩阵，其转置等于其逆，因此二维 DCT 变换的逆变换也可以用类似的结构表达：

$$f = A^{-1}F(A^T)^{-1} = A^TFA$$

相比基于快速傅立叶变换的做法，本方法缺点是因为涉及矩阵乘法，时间复杂度较高（O(n^3)）；优点是易于理解，且借助 OpenCV 重载的矩阵乘法，实现非常简便。

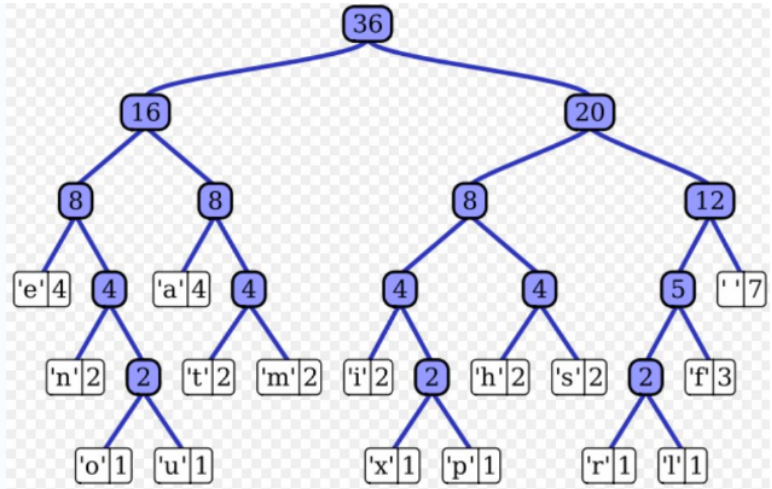
### 3.1.2 游程编码

游程编码的核心思想是“使用变动长度的码来取代连续重复出现的原始资料”来实现压缩。

举例来说，"AAAABBBCCDEEEE"，由 4 个 A、3 个 B、2 个 C、1 个 D、4 个 E 组成，经过 RLE 压缩后输出为 4A3B2C1D4E，由 14 个字符缩减到 10 个字符。对重复性较高的数据，RLE 游程编码非常有效。

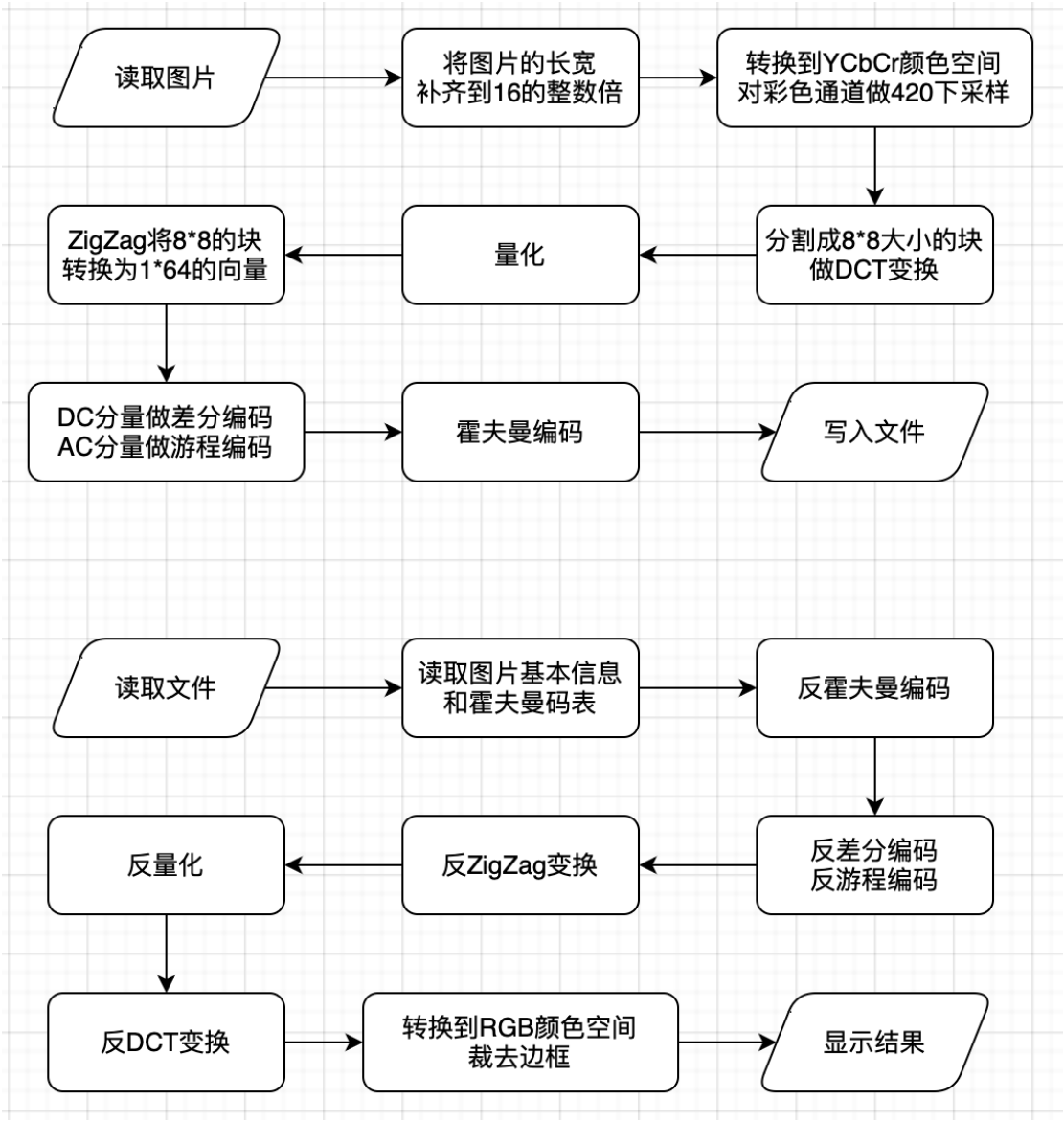
### 3.1.3 霍夫曼编码

霍夫曼编码是熵编码的一种，它的基本思想是根据字符出现的频率来构建一个最优的前缀编码表。在编码时，霍夫曼编码将出现频率较高的字符用较短的编码表示，出现频率较低的字符用较长的编码表示，从而实现数据压缩。



## 3.2 代码实现

### 3.2.1 整体流程




本算法完整遵循了标准 JPEG 使用的压缩步骤，区别主要在于：

1. 省略了众多控制字段，最终输出文件的格式不同。
2. 标准 JPEG 中三个通道各自的 AC、DC 分量分别进行霍夫曼编码，因此有六个数据段，六个码表；而本算法中三个通道共用一个码表。

### 3.2.2 长宽补齐到 16 的倍数

读取图片后第一步处理是将其长宽向上补齐到 16 的倍数，这是为了保证经过 4:2:0 颜色下采样后，颜色通道的长宽仍然可以被 8 整除。本步骤使用 OpenCV 提供的 CopyMakeBorder 函数完成。值得注意的是，本步骤中边框填充方案选用

的是 BORDER_REFLECT，即填充的部分与原图关于图片原有边缘镜像对称。这是为了让填充的部分的信息相比原图对称相似，从而有利于后续的 DCT 变换得到较为规整的结果。

```
copyMakeBorder( src_img, src_img,
                0, bottom_border, 0, right_border,
                BORDER_REFLECT);
```

最终输出的文件中包含未经填充的原图尺寸。在解码最开始，解码器会根据原图尺寸用相同的算法得到填充后的尺寸，并以此进行解码；在解码的末尾，解码器会将填充的边框裁去再进行显示。

### 3.2.3 色彩空间转换

OpenCV 图片默认以 BGR 格式读取，而在后续压缩中需要使用 YCbCr 通道，这里没有使用 OpenCV 自带的转换函数，而是通过 ITU.BT-601 标准提供的公式手动计算：

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.000 & 1.403 \\ 1.000 & -0.344 & -0.714 \\ 1.000 & 1.773 & 0.000 \end{bmatrix} \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix}$$

### 3.2.4 颜色下采样

本算法中对图片使用 4:2:0 颜色下采样。代码利用了 OpenCV 提供的 Resize 函数，实现方式比较取巧：将彩色通道矩阵的长宽均缩小到原来的一半，每个 2*2 田字形区域内的值取线性平均（INTER_LINEAR）作为新的值。

```
resize( channel, channel,
        Size(channel.cols / 2, channel.rows / 2), 0, 0,
        INTER_LINEAR);
```

与之对应的，解码器中将缩小过后的矩阵长宽均放大一倍，每个像素的值会被复制 4 份（INTER_NEAREST）并填放在其相邻的 2*2 区域中。

```
resize( channel, channel,
        Size(channel.cols * 2, cb_channel.rows * 2), 0, 0,
        INTER_NEAREST);
```

### 3.2.5 DCT 变换

与 JPEG 一样，本算法以 8*8 的块为 DCT 的基本单位。实现 DCT 变换与反变换的方式在 3.1 节中已有详细介绍，这里不做赘述。

$$A(i,j) = c(i)cos[\frac{(j+0.5)i\pi}{N}]$$

$$f = A^{-1}F(A^T)^{-1} = A^TFA$$

$$F = AfA^T$$

### 3.2.6 量化

本算法使用 JPEG 标准中定义的量化矩阵。下图中左侧是亮度通道的量化矩阵，右侧是颜色通道的量化矩阵。

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

### 3.2.7 Zigzag

对前一步量化的结果（8*8 矩阵）进行编码，顺序是左上斜对角到右下，而不是逐行或者逐列。这是为了 DC 与低频的 AC 信号放在一起，方便后续的编码。

本算法中使用查表方式进行 Zigzag 变换。按行从 0 到 63 为矩阵内元素编号，如下方左侧所示；然后进行 Zigzag 扫描，输出向量下标与矩阵内元素编号的对应关系如下方右侧所示。

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix} \begin{bmatrix} 0 & 1 & 8 & 16 & 9 & 2 & 3 & 10 \\ 17 & 24 & 32 & 25 & 18 & 11 & 4 & 5 \\ 12 & 19 & 26 & 33 & 40 & 48 & 41 & 34 \\ 27 & 20 & 13 & 6 & 7 & 14 & 21 & 28 \\ 35 & 42 & 49 & 56 & 57 & 50 & 43 & 36 \\ 29 & 22 & 15 & 23 & 30 & 37 & 44 & 51 \\ 58 & 59 & 52 & 45 & 38 & 31 & 39 & 46 \\ 53 & 60 & 61 & 54 & 47 & 55 & 62 & 63 \end{bmatrix}$$

这个对应关系不会改变，程序中中硬编码了上面右侧的表格，Zigzag 只需要查表做下标变换即可。

```
// Encode
for (i in 0…8)
    for (j in 0…8)
```

```
        dst_vec[ lut[i][j] ] := src_mat[i][j]
// Decode
for (i in 0…8)
    for (j in 0…8)
        src_mat[i][j] := dst_vec[ lut[i][j] ]
```

### 3.2.8 差分编码

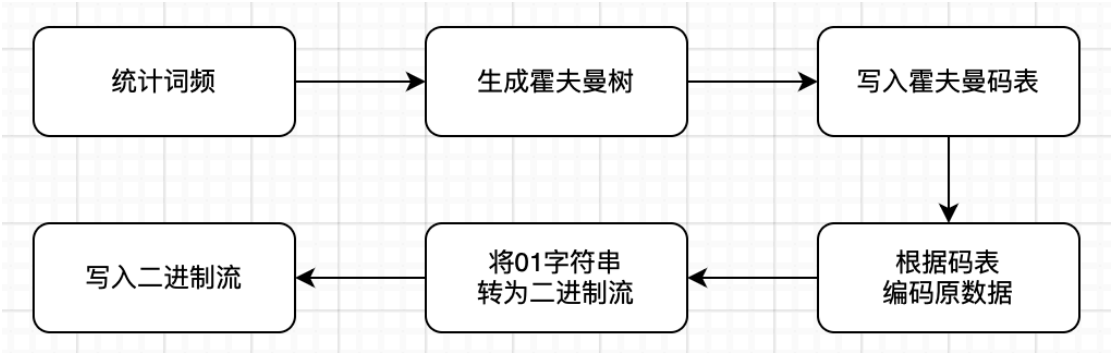
对 DC 分量，即 Zigzag 输出向量中下标为 0 的元素，使用基础的差分编码。

```
data[i] := dc[i] – dc[i – 1]
```

### 3.2.9 游程编码

RLE 游程编码的原理在 3.1 节中已有说明，这里不做赘述。

### 3.2.10 霍夫曼编码




出于实现复杂度与空间利用率的权衡，所有数据共同生成、使用一张霍夫曼码表，码表使用明文写入文件。生成霍夫曼树的算法在 3.1 节中有详细介绍，这里不再重复

C 语言最小支持以字节为单位向文件中写入数据，但是 01 字符串的长度不一定能够被 8 整除，在编码转为二进制流时需要补 0；因此在每一个数据块的头部，需要额外记录 01 字符串的原始长度。

解码时，根据任意霍夫曼编码不可能是其他编码前缀的原理，每次读入一个比特，指针便指向重建的霍夫曼树的左或右孩子；如果指针指向的节点没有后代节点，则读取该节点对应的字符。同时，每读入一个比特，都需要记录已读的数据量，并与总长度进行比对，以防将末尾填充的 0 误作为有意义编码读入。

# 4. Experiment Results

## 4.1 用户界面

可执行文件在命令行环境下运行。编码器可执行文件名为 encode2708，使用方法如下，其中源文件可以是任意 OpenCV 支持打开的图片格式：

> $ ./encode2708 [source picture path] [saved file name]

解码器可执行文件名为 decode2708，使用方法如下：

> $ ./decode2708 [source file path]

如参数不全会有提示信息，运算过程中会实时输出当前进度：

```
> ./encode2708 case1.png case1.out
Source file: case1.png does not exist.

> ./encode2708 case1.jpg case1.out
Start compress:
        Channel separation
        Color subsampling
        DCT
        Quantification
        Zigzag serialization
        DPCM & RLC
        Huffman encoding
> ./decode2708 case1.out
Start decompress:
        Huffman decoding
        Inverse DPCM & RLC
        Zigzag deserialization
        Inverse quantification
        Inverse DCT
        Picture reconstruction
```

解码成功后会新建窗口显示图片。

## 4.2 效果分析

项目随附了 5 个测试样例，他们的基本信息如下：

| 名称 | 分辨率 | 位图大小 | 原格式 | 原大小 |
|---|---|---|---|---|
| case1 | 1920×2560 | 14,745KB | jpg | 1,018KB |
| case2 | 3456×5184 | 53,748KB | jpg | 7,863KB |
| case3 | 480×789 | 1,136KB | png | 957KB |
| case4 | 3024×4032 | 36,578KB | png | 15,134KB |
| case5 | 5504×6421 | 106,024KB | jpg | 8,682KB |

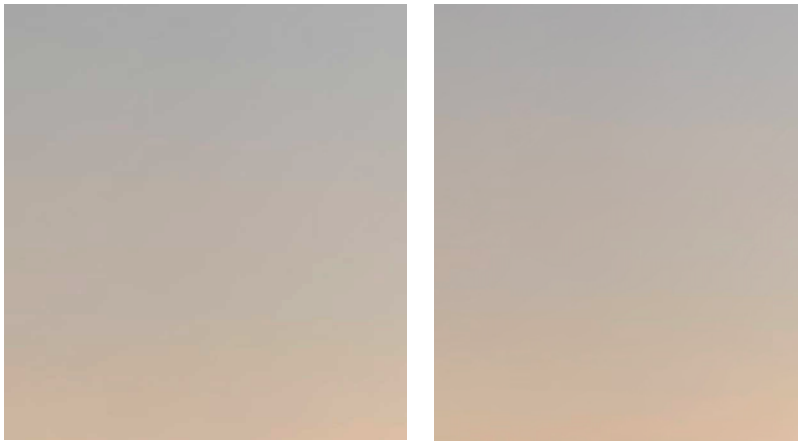
首先测试本算法的压缩效率：

| | case1 | case2 | case3 | case4 | case5 |
|---|---|---|---|---|---|
| **压缩后大小** | 393KB | 1,085KB | 166KB | 1,838KB | 3,882KB |
| **压缩率（原图）** | 38.6% | 13.8% | 17.3% | 12.0% | 44.7% |
| **压缩率（位图）** | 2.7% | 2.0% | 14.6% | 5.0% | 3.7% |

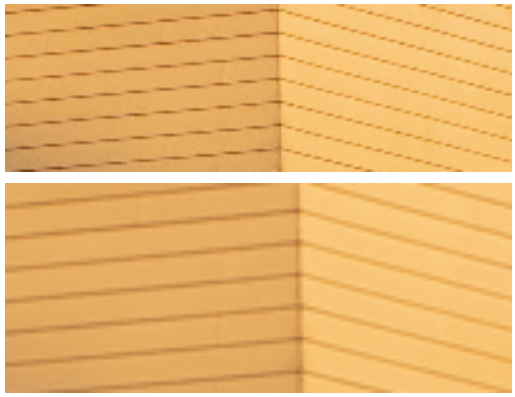
由上表可见，算法的压缩效率比较理想，对比效率较高的 JPEG 格式，压缩率仍然达到了约 40%；对比效率较低的 JPEG 和 PNG 格式，压缩率则进一步提高到 15%左右；对比效率最低的位图，压缩率则普遍在 10%以下，针对有大面积单一颜色的 case2（证件照），压缩率更是达到了 2%，可见 DCT、RLC 等算法作用之显著。

下面分析对比原图画质损失较为明显的两处：

1. 下图左侧是 case1 用本算法压缩后天空，对比右侧原图出现了色彩断层。色彩断层在 JPEG 压缩中是一种非常常见的现象，主要原因是颜色的量化粒度较大，不能很好的保留过渡平缓的渐变色。



2. 下图上方是 case5 经本算法压缩后的风车线条，对比下方原图出现了比较明显的锯齿，主要原因是下采样时将颜色值 4 合 1 会破坏斜线原本平滑的边缘，而 case5 中风车上平行且细密的线条使这种现象尤为明显。



除上面两处之外，其余部分画质损失基本不可查，考虑到压缩效果较为显著，我认为这样的画质损失是可以接受的。

## References:


参考网址：

https://www.jianshu.com/p/b8fa4b368712

https://zh.wikipedia.org/wiki/游程编码

https://github.com/Harryline-996/MATLAB_for_JPEG_compression

https://app.diagrams.net

https://blog.csdn.net/mjx91282041/article/details/13020543

https://zhuanlan.zhihu.com/p/498289926

https://zhuanlan.zhihu.com/p/28766366

https://blog.csdn.net/crayfish104/article/details/118031793

特别感谢：

https://chat.openai.com/