浙江大学

课程名称: 多媒体技术

指导老师: 肖俊

完成时间: 2023 年 5 月 13 日

序号	姓名	学号	班级	性别	分工
1				男	代码
2				男	代码审查，报告

图像压缩和解压系统

一、项目介绍

1.1 总体信息

项目内容：JPEG 压缩、解压算法

项目实现：

1. 不同图片格式转换为 JPEG 格式。
2. JPEG 压缩算法全过程：
   - RGB 色彩空间到 YUV 转换；
   - 对图像块进行 2D-DCT 变换；
   - 使用量化矩阵对 DCT 系数进行调整；
   - 进行 Zigzag 排序便于压缩高频系数；
   - 对 DC 系数进行 DPCM；
   - 对 AC 系数进行 RLE；
   - 通过 Huffman 进行熵编码。
3. 压缩后码流还原图片文件。

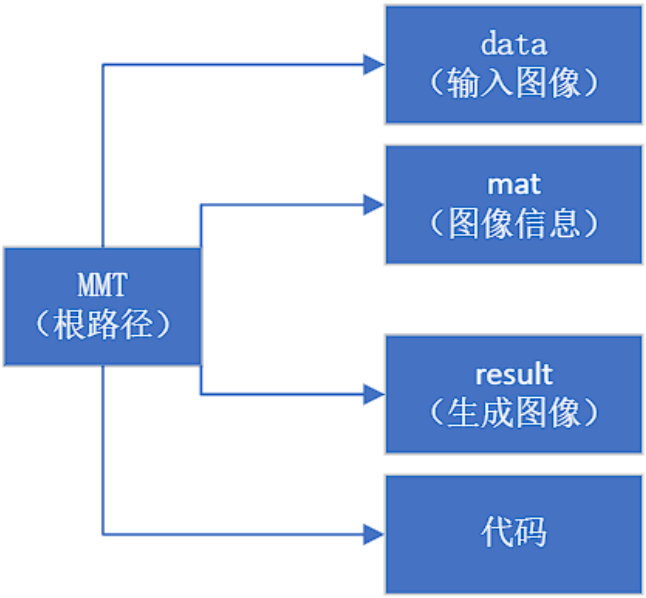
项目开发环境：

操作系统：win10

语言版本：matlab R2020b

1.2 项目结构

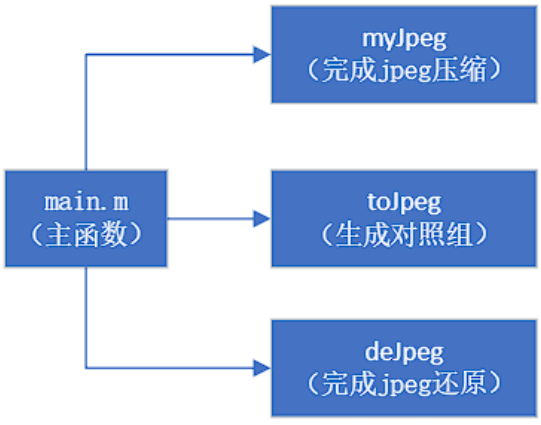
1.2.1 项目路径结构



1.2.2 项目代码结构

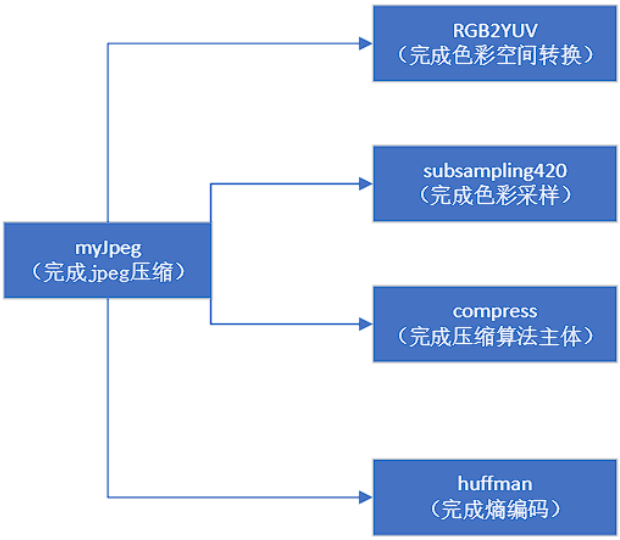
1. 项目主函数结构

项目的主函数调用了三个函数，其功能如下：



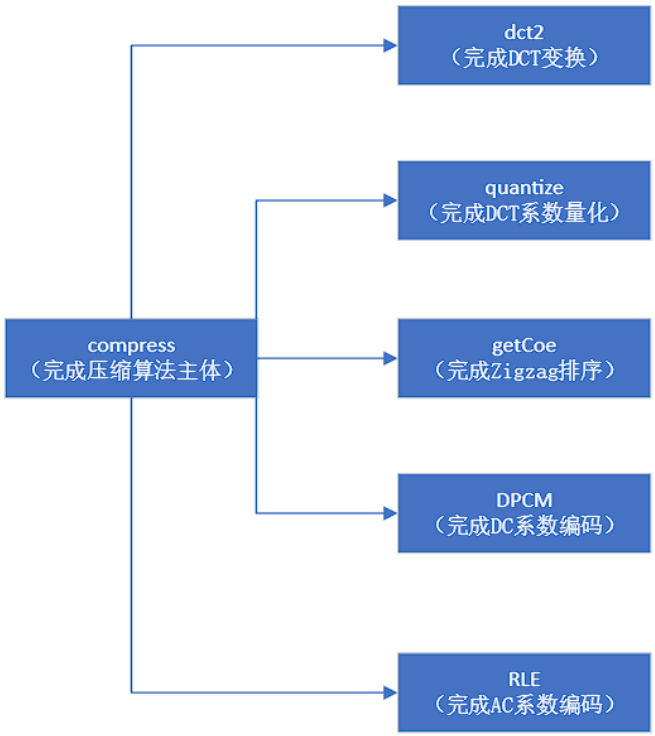
2.myJpeg 压缩函数结构

myJpeg 实现了其他图像文件转换为 JPEG 文件的压缩处理：



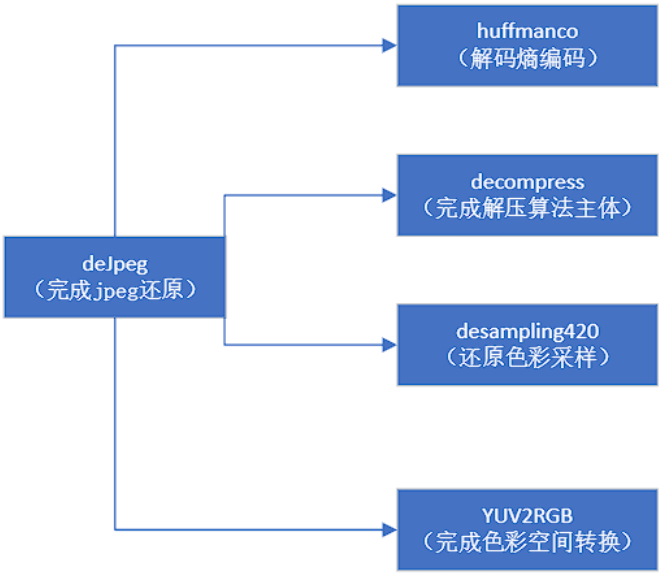
3.compress 函数结构

compress 函数是 myJpeg 压缩函数最重要的部分，实现了压缩算法的核心内容：



4.deJpeg 函数结构

deJpeg 函数可以基于被压缩的 JPEG 文件与 mat 文件夹的图像信息还原原格式的图像文件，其代码结构如下：



二、技术实现

2.1 技术原理

2.1.1 色彩空间转换

虽然项目内的文件名是 RGB2YUV，但是本项目实际上使用 YCbCr 色彩空间进行编码。RGB 到 YCbCr 色彩空间转换是指将 RGB 颜色模型中的三个分量（红、绿、蓝）转换为 YCbCr 颜色模型中的三个分量（亮度 Y、蓝色偏移量 Cb、红色偏移量 Cr）。其转换公式是：

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \tag{1}$$

$$Cb = -0.169 * R - 0.331 * G + 0.499 * B + 128 \tag{2}$$

$$Cr = 0.499 * R - 0.439 * G - 0.081 * B + 128 \tag{3}$$

2.1.2 色彩采样

色彩采样是一种色彩压缩的方法，它可以减少信号中的色彩信息，而保留亮度信息。这样可以降低带宽占用，而不会明显影响画质。

一个视频信号可以分为两个部分：亮度信息和色彩信息。亮度信息，或者叫做亮度分量（Y），决定了画面的大部分细节，因为对比度是我们看到屏幕上形状的主要因素。例如，一张黑白照片并不会比一张彩色照片看起来更模糊。色彩信息，或者叫做色度分量（Cb 和 Cr），也很重要，但是对视觉影响较小。

4:2:0 色彩采样的具体方法是，在每四个像素中，只对其中一个像素进行色度采样，而忽略其他三个像素的色度信息。这样就相当于将水平和垂直方向的色度分辨率都降低了一半。例如，在一个 4x2 的像素阵列中，只有左上角的那个像素有完整的 YCbCr 信息，其他像素只有 Y 信息。

2.1.3 2D-DCT 与 quantize

2D-DCT 变换是二维离散余弦变换的简称，它是一种将图像从空间域转换到频域的方法，可以有效地压缩图像的冗余信息，提高图像的压缩比和质量。

2D-DCT 变换的基本思想是将一个 N×N 的图像分块为 n×n 的子块（通常 n=8），然后对每个子块进行一维离散余弦变换（DCT），得到 n 个一维 DCT 系数，再对这些系数进行一维 DCT，得到 n×n 个二维 DCT 系数。

2D-DCT 变换的常见公式如下：

$$F(u,v) = fracC(u)C(v)sqrt2nsum_{x=}^{n}{}_{=}^{1}sum_{y=}^{n}{}_{=}^{1}f(x,y)cosfrac(2x+1)upi2ncosfrac(2y+1)vpi2n$$

在本次项目中我们使用子块大小为 8*8，因此：

$$F(u,v) = fracC(u)C(v)4sum_{x=0}^{7}sum_{y=0}^{7}f(x,y)cosfrac(2x+1)upi16cosfrac(2y+1)vpi16$$

DCT 系数的量化是一种将 DCT 系数的幅值降低，增加零系数的个数，从而实现图像压缩的方法 123。量化的原理是把变换后的 DCT 系数除以一个常量，经过量化后的结果是量化步长的整数倍或者为更多的零值。

本次使用量化表法：根据人眼对不同空间频率敏感程度不同，设计一个 8×8 的量化表，其中每个元素表示对应位置的 DCT 系数的量化步长。这种方法可以灵活地调整量化表中的元素值，以适应不同图像和质量要求。

2.1.4 DPCM 与 RLE

DPCM 是一种利用相邻图像块之间 DC 系数的差值进行编码的方法。它的基本原理是：

* 对于每个图像块，取左上角的 DCT 系数作为 DC 系数，这个系数表示图像块的平均亮度。

* 对于第一个图像块，直接输出其 DC 系数作为编码结果。

* 对于后续的图像块，计算其 DC 系数与前一个图像块的 DC 系数的差值，输出这个差值作为编码结果。

RLE 的全称是游程长度编码，它的基本原理是：

* 对于每个 8×8 的图像块，取除了左上角的 DCT 系数之外的其他 63 个系数作为 AC 系数，这些系数表示图像块的细节变化。

* 对于每个图像块，按照 Z 字形扫描的顺序，将 AC 系数排列成一个一维向量。

* 对于每个一维向量，统计其中连续的零的个数，写入。紧跟着是下一个非零系数的值。

* 如果一维向量以零结尾，则用一个特殊的符号 EOB（End Of Block）来表示后面都是零。如果一维向量不以零结尾，则不需要 EOB。

在本实验中，为了代码编写方便，全部使用 EOB 符号。

RLE 可以压缩掉 AC 系数中大量的冗余信息，因为量化后的 AC 系数往往包含很多连续的零。

RLE 可以利用 Huffman 编码进一步压缩数据，因为 RLE 生成的字节和非零系数有一定的概率分布，可以根据概率分配不同长度的编码。

2.1.5 Huffman 编码

Huffman 编码是一种利用字符出现的概率来构造变长编码的方法。

Huffman 树的构建方法是：将所有像素值按照权值升序排序，每次取权值最小的两个节点合并成一个新节点，新节点的权值为两个子节点的权值之和，然后将新节点插入到排序队列中，重复这个过程直到只剩下一个根节点。

Huffman 编码的生成方法是：从根节点开始往下走，向左走为 0，向右走为 1。每个像素值对应的编码就是根节点到该像素值所在节点的路径编码。这样可以保证每个编码都不是其他编码的前缀，便于解码。

2.2 算法实现

2.2.1 色彩空间转换

应用 3.1.1 中色彩转换公式。

RGB 到 YCbCr 转换代码如下：

```
ret(i, j, 1) = 0.299 * image(i, j, 1)
  + 0.587 * image(i, j, 2) + 0.114 * image(i, j, 3);
ret(i, j, 2) = -0.1687 * image(i, j, 1)
  - 0.3313 * image(i, j, 2) + 0.5 * image(i, j, 3) + 128;
ret(i, j, 3) = 0.5 * image(i, j, 1)
  - 0.4187 * image(i, j, 2) - 0.0813 * image(i, j, 3) + 128;
```

同理取逆运算，YUV 到 RGB 转换代码如下：

```
ret(i, j, 1) = image(i, j, 1)
  + 1.402 * (image(i, j, 3) - 128);
ret(i, j, 2) = image(i, j, 1) - 0.34414 * (image(i, j, 2) - 128)
```

```
  - 0.71414 * (image(i, j, 3) - 128);
ret(i, j, 3) = image(i, j, 1) + 1.772 * (image(i, j, 2) - 128);
```

2.2.2 色彩采样

色彩采样代码需要根据 Y、Cb、Cr 进行差异化处理：

```
imageY = image(:, :, 1);
imageU = subsampling420(image(:, :, 2));
imageV = subsampling420(image(:, :, 3));
```

对于 Cb 和 Cr 两个色度分辨率较低的维度进行 4:2:0 下采样：

```
function [ret] = subsampling420(matrix)

    [sizeX, sizeY] = size(matrix);
    sizeX = cast(sizeX / 2, "int32");
    sizeY = cast(sizeY / 2, "int32");

    ret = zeros([sizeX, sizeY]);
    for i = 1 : sizeX
        for j = 1 : sizeY
            ret(i, j) = matrix(2 * i - 1, 2 * j - 1);
        end
    end
end
```

2.2.3 2D-DCT 与 quantize

本次项目中 2D-DCT 处理使用了工具箱中的 dct2 函数，之后按照 AC 和 DC 分量分别进行量化：

```
cur = dct2(cur);
cur = quantize(cur, arg);
```

其中的量化步骤：

```
function [ret] = quantize(matrix, arg)

```

```
    quanMat = getConstMatrix(arg);
    ret = cast(round(matrix ./ quanMat), 'int32');
end
```

2.2.4 DPCM 与 RLE

DPCM 编码输出 DC 系数与前一个图像块的 DC 系数的差值，因此代码实现如下：

```
function [ret] = DPCM(input)

    ret = input;
    len = length(input);

    for i = 2 : len
        ret(i) = input(i) - input(i - 1);
    end
end
```

RLE 的实现如下：

```
function [ret] = RLE(input)

    len = length(input);
    fir = 0;
    ret = [];
    for i = 1 : len
        if input(i) == 0
            fir = fir + 1;
        else
            ret = [ret, fir, input(i)];
            fir = 0;
        end
    end

    ret = [ret, fir, 0];
end
```

2.2.5 Huffman 编码

Huffman 关键在于 Huffman 树的生成。鉴于可以使用 MATLAB 库的 huffman 相关函数，可以只进行字典和词频统计，而让 matlab 进行编码。代码如下：

```
function [dict, code] = huffman(input)

    len = length(input);
    alpha = [];
    prob = [];
    for i = 1 : len
        flag = 0;
        lenD = length(alpha);
        for j = 1 : lenD
            if alpha(j) == input(i)
                flag = j;
                break;
            end
        end
        if flag ~= 0
            prob(flag) = prob(flag) + 1;
        else
            alpha = [alpha, input(i)];
            prob = [prob, 1.0];
        end
    end

    lenD = length(alpha);
    for i = 1 : lenD
        prob(i) = prob(i) / len;
    end

    dict = huffmandict(alpha, prob);
    code = huffmanenco(input, dict);
end
```

这里得到的 dict 和 code 经过进一步的位压缩处理，就可认为是我们最后得到的压缩信息。

三、运行结果

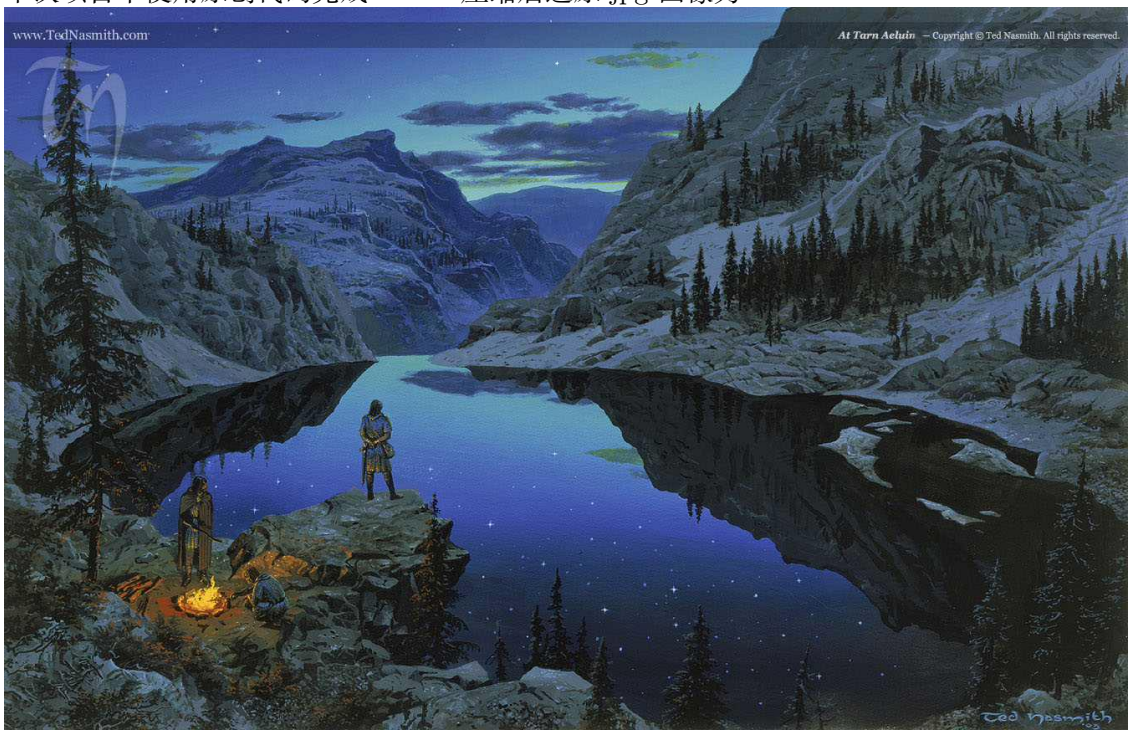
3.1 输入输出图像对比

不妨取 Aeluin.jpg 作为范例展示本次程序的运行：

源文件图像为：

toJPEG 使用 matlab 库函数自动生成的对照 JPEG 如下：

本次项目中使用原创代码完成 JPEG 压缩后还原 jpg 图像为：



从图像表现可知，本次 JPEG 压缩、还原算法实现较为成功，在没有过度影响肉眼观感的情况下完成了任务。

但同时，放大后可以发现，原创压缩也牺牲了一些细节，尤其在 Yemen.png 中较为明显：

在源文件中，白色图形与红色背景之间的界限清晰：

而经过原创压缩之后，白色和红色之间的界限之间出现了颜色改变、锯齿等细节丢失的现象：

这是因为 JPEG 压缩是有损压缩，色度采样、DCT 系数量化等操作都会导致信息损失，尤其是在不同对象之间的界限处，因为那里的高频信息更多，所以压缩后的图片会出现一些失真和模糊的现象。

3.2 压缩率计算

本次实验一共统计了五个不同格式图像样本源文件和 MATLAB 标准库函数压缩、原创压缩之后的压缩率对比：

文件名	原字节数	官方压缩字节数	压缩率 1	原创压缩字节数	压缩率 2
Aeluin.jpg	298,209	216,309	1.379	184,501	1.616
RGBB.tif	322	661	0.487	343	0.939
Valinor.jpg	264,850	202,182	1.301	170,692	1.552
Wallachia.png	72,228	46,361	1.558	29,967	2.410
Yemen.png	18,296	19,585	0.934	11,724	1.561

对比压缩率 1 和压缩率 2 可知，本次项目原创实现的 JPEG 压缩比起 MATLAB 标准库函数压缩

效果要好，压缩率更大。

本次实验中，Wallachia.png 获得了最大的压缩比，是因为原图是白底，对 Cb 和 Cr 值进行 RLE 编码能极度减少储存信息。

RGBB.tif 是一个 8*8 的用于调试的小图像。对这个图像的压缩，无论是 MATLAB 标准库压缩和原创压缩都不太理想，原创压缩由于量化表系数的原因，甚至丢失了 Cb 和 Cr 信息，导致最后生成了一个灰度图像。

此外，由于实际上本实验未直接写成二进制码流文件，而是采用 MATLAB 的 save 和 load 直接进行编码后向量的储存和读取，压缩比实际上还有较大的改进空间。

四、参考材料

多媒体技术课程及教材

DCT：https://en.wikipedia.org/wiki/Discrete_cosine_transform

Huffman：https://en.wikipedia.org/wiki/Huffman_coding