信息论 project

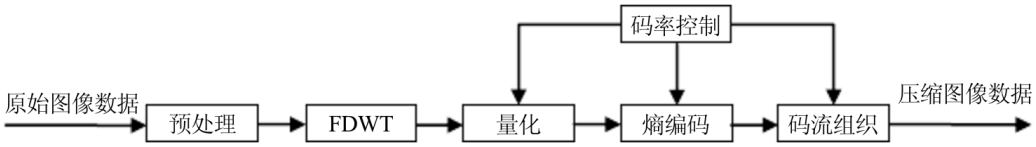
JPEG2000

高云帆 15307130314
张奕朗 16307130242
陈幸豪

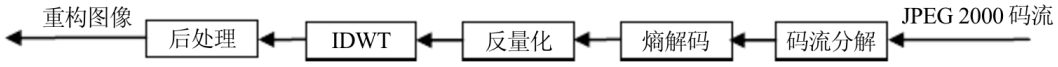
2018 年 12 月 19 日

1 综述

JPEG 2000 是基于小波变换的图像压缩标准，由 Joint Photographic Experts Group 组织创建和维护．本次 project 的目的是复现 JPEG 2000 的压缩流程，完成一个“自定义”的 JPEG 2000 压缩程序．其核心编解码模块与过程如下图所示：



(a) 编码过程



(b) 解码过程

图 1.1　JPEG 2000 核心编解码模块与过程.

小组分工如下：

张奕朗：　图像分块，电平平移归一化，小波变换

陈幸豪：　色彩分量变换，量化

高云帆：　EBCOT（最佳截断嵌入码块编码），算术编码，数据流

2 图像分块与电平平移归一化

在图像变换之前，首先需要对图像进行预处理．预处理主要分为分块、电平平移归一化和颜色变换三个过程（**图2.1**）．在调用 *cv2.imread* 读取图像数据时，利用正则表达式获取图像的颜色位数信息，方便之后的处理．*代码见函数 init_image*.

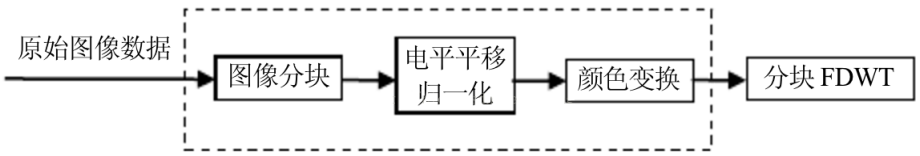


图 2.1　图像预处理.

2.1 图像分块

与 JPEG 标准相同，JPEG2000 标准也需要将图像分块，不过是为了降低计算量和对内存的需求，以方便压缩（当然也可以选择不分块）．但与 JPEG 标准不同的是，JPEG2000 标准并不需要将图像强制分成 8×8 的小块, 而是可以将图像分割成若干任意大小的互不重叠的矩形块 (tile), 常分为 $2^6 \sim 2^{12}$（即 64 ~ 1024 像素宽）的正方形 tile. 受图像形状的影响，边缘部分的 tile 的大小和形状可能与其它的 tile 不同．Tile 的大小会影响重构图像的质量．一

般情况下，大的 tile 重构出的图像质量要好一些，因为小波变换的范围大，减轻了边缘效应. 代码见 *class Tile* 和 *class JPEG2000* 中的函数 *image_tiling*.

2.2 电平平移归一化

电平平移归一化分为两步：

第一步是直流电平平移（DC level shifting），通过将数据减去均值使之关于 0 对称分布，以去掉频谱中的直流分量.

$$I_1(x,y) = I(x,y) - 2^{B-1}$$

$$-2^{B-1} \le I_1(x,y) \le 2^{B-1} - 1$$

其中 B 是之前读取到的颜色位数信息.

第二步是电平归一化（normaliztion）. 对于无损压缩中的 5/3 小波变换，由于采用的是整数小波变换，所以不需要进行归一化；而对于有损压缩中的 9/7 小波变换，由于采用的是实数小波变换，故需要对每个 tile 进行如下运算以归一化：

$$I_2(x,y) = \frac{I_1(x,y)}{2^B}$$

$$-\frac{1}{2} \le I_2(x,y) < \frac{1}{2}$$

但是事实上，电平平移归一化对 DWT 来说不是必须的（而且电平平移归一化后还不便于进行色彩空间变换），因为它只会影响小波系数的动态范围而不会影响结果. 代码见函数 *dc_level_shift*.

2.3 反变换

在解码、小波反变换和色彩空间反变换之后，需要进行电平反平移，（如果是有损压缩，还需要进行反归一化），最后拼接得到恢复的图像. 这些都是上述操作的反过程，这里不再赘述. 代码见函数 *idc_level_shift* 和 *image_splicing*.

3 小波变换

3.1 小波变换原理简述

对非平稳过程，傅里叶变换有其局限性. 它只能获取一段信号总体上包含哪些频率的成分，但是对各成分出现的时刻并无所知. 因此时域相差很大的两个信号，可能频谱图一样. 而加窗截取信号的 STFT（短时傅里叶变换）又存在窗长的问题：窗太窄，频率分辨率差；窗太宽，时间分辨率低. 于是便产生了利用有限长的会衰减的小波基进行与源信号进行相关的 WT（小波变换）和 DWT（离散小波变换），需要用到的 DWT 的变换和逆变换公式如下：

$$WT[j,k] = \sum_n \langle f, \psi_{j,k} \rangle = \frac{1}{\sqrt{j}} \sum_n f[n] \psi[\frac{n-k}{j}]$$

$$f[n] = \sum_{j,k} \langle f, \psi_{j,k} \rangle \hat{\psi}_{j,k} = \frac{1}{A} \sum_{j,k} WT[j,k] \psi_{j,k}$$

不同于傅里叶变换，变量只有频率 ω，小波变换有两个变量：尺度 j 和平移量 k. 尺度 j 控制小波函数的伸缩，平移量 k 控制小波函数的平移. 尺度 j 对应于频率（反比），平移量 k 对应于时间.

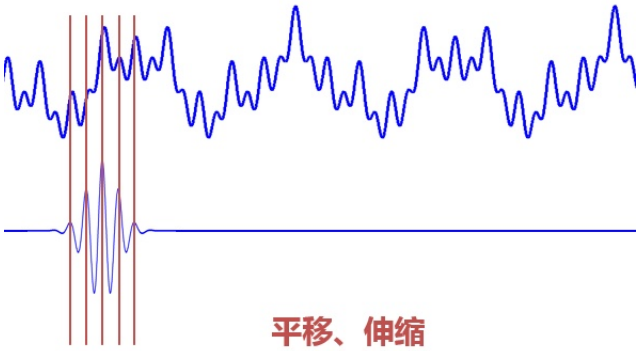


图 3.1　小波基的尺度参数和平移参数.

3.2　小波变换的滤波器实现与提升算法

小波变换的滤波器等效

小波变换可以用 IIR 滤波器进行等效（具体推导见《JPEG2000 图像压缩基础、标准和实践》），并且可以通过提升算法进行实现. 在有损压缩的情况下，JPEG2000 标准的核心编码系统默认的不可逆小波变换是 Daubechies 9/7 DWT 的提升实现，而在无损情况下则采用 Le Gall 5/3 滤波器的提升实现的整数可逆小波变换. 其中 Daubechies 9/7 是 I.Daubechies 与 M.Antonini 等人于 1992 年提出的一种双正交小波滤波器. Le Gall 5/3 是 D.Le Gall 与 A.Tabatabai 于 1988 年基于样条 5/3 变换而提出的一种可逆双正交滤波器. 两者的滤波器参数见**图??**、**??**. 其中参数 9/7 和 5/3 分别指分解低/高通滤波器的 IIR 长度，合成滤波器相反（正交对称）.

	分解滤波器系数		合成滤波器系数	
	低通滤波器 $h_L(i)$	高通滤波器 $h_H(i)$	低通滤波器 $g_L(i)$	高通滤波器 $g_H(i)$
0	0.6029490182363579	1.115087052456994	1.115087052456994	0.6029490182363579
±1	0.2668641184428723	-0.5912717631142470	0.5912717631142470	-0.2668641184428723
±2	-0.07822326652898785	-0.05754352622849957	-0.05754352622849957	-0.07822326652898785
±3	-0.01686411844287495	0.09127176311424948	-0.09127176311424948	0.01686411844287495
±4	0.02674875741080976	0	0	0.02674875741080976
其他值	0	0	0	0

图 3.2　Daubechies 9/7 滤波器系数表.

i	分解滤波器系数		合成滤波器系数	
	低通滤波器 $h_L(i)$	高通滤波器 $h_H(i)$	低通滤波器 $g_L(i)$	高通滤波器 $g_H(i)$
0	6/8	1	1	6/8
±1	2/8	-1/2	1/2	-2/8
±2	-1/8	0	0	-1/8
其他值	0	0	0	0

图 3.3　Le Gall 5/3 滤波器系数表.

小波变换的提升算法

Daubechies 9/7　正变换包括 4 步提升和 2 步缩放：

4 步提升：

$$y(2n+1) = x(2n+1) - \alpha[x(2n) + x(2n+2)]$$
$$y(2n) = x(2n) - \beta[y(2n-1) + y(2n+1)]$$
$$y(2n+1) = y(2n+1) + \gamma[y(2n) + y(2n+2)]$$
$$y(2n) = y(2n) + \delta[y(2n-1) + y(2n+1)]$$

2 步缩放：

$$y(2n+1) = -K\,y(2n+1)$$
$$y(2n) = y(2n)/K$$

逆变换包括 2 步缩放和 4 步提升：

2 步缩放：

$$x(2n+1) = -y(2n+1)/K$$
$$x(2n) = K\,y(2n)$$

4 步提升：

$$x(2n) = x(2n) - \delta * [x(2n-1) + x(2n+1)]$$
$$x(2n+1) = x(2n+1) - \gamma * [x(2n) + x(2n+2)]$$
$$x(2n) = x(2n) + \beta * [y = x(2n-1) + x(2n+1)]$$
$$x(2n+1) = x(2n+1) + \alpha * [x(2n) + x(2n+2)]$$

其中 $\alpha, \beta, \gamma, \delta, K$ 为与滤波器系数有关的参数：$\alpha = 1.586134342, \beta = 0.052980118, \gamma = 0.882911075, \delta = 0.443506852, K = 1.230174105$.

Le Gall 5/3 正、逆变换均只需要一步提升与一步缩放.

正变换：

$$y(2n+1) = x(2n+1) - [x(2n) + x(2n+2)]/2$$
$$y(2n) = x(2n) + [y(2n-1) + y(2n+1) + 2]/4$$

逆变换：

$$x(2n) = y(2n) - [y(2n-1) + y(2n+1) + 2]/4$$
$$x(2n+1) = y(2n+1) + [x(2n) + x(2n+2)]/2$$

3.3 二维多级小波变换

对图像进行 2D-DWT 后可得到图像的 2D-DWT 系数，将系数按照左上、左下、右上、右下分别分为 LL、LH、HL、HH 四个分量（L 表示低频，H 表示高频）. 由于图像的信息大多集中在低频区域，所以若分解所得的 LL 分量仍含有较多的图像信息时（动态范围大，不利于量化和编码），可将其进一步做 2D-DWT 分解，也就是多级小波分解，如**图??**所示. JPEG2000 中一般做 3 ∼ 4 级 2D-DWT，经实践，选用 3 级 2D-DWT 就已经足够了.

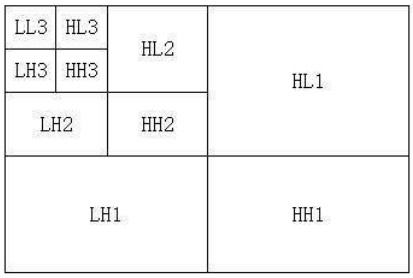


图 3.4　二维小波变换的多级划分.

开始时我用 *for* 循环实现了提升算法，但是运行速度太慢，于是改为调用 *python* 的 PyWavelets 包，建立自定义小波来实现 2D-DWT. 虽然 Daubechies 9/7 是以 Daubechies 命名的，但它和普通的 DB 族小波基都不同：普通的 DB 小波基是非对称非正交小波基，而 Daubechies 9/7 是双正交对称小波基，在查阅官方的小波库文档之后，重新按照双正交对称小波基的参数进行排列，最终实现了基于 DB 9/7（有损情况）和 LG 5/3 滤波器（无损情况）的 2D-DWT 和 2D-iDWT，且速度较原来 *for* 循环的算法有了很大的提升. 代码见函数 *dwt* 和 *idwt*.

4　内嵌区段编码

4.1　基本编码算法

零编码（zero coding）

该编码根据待编码的数据比特周围的 8 个相邻数据重要性情况生成上下文（CX）. 如表 **??**所示，D0、D1、D2、D3 分别表示待编码数据比特 X 对角线的数据状态值；V0、V1 分别表示 X 垂直方向的状态值，H0、H1 分别表示 X 平方向的状态值. 通过计算 $\sum H_i, \sum V_i, \sum D_i$ 的数值，对 X 进行基于子带的编码. 返回 D 和 Context，D 为 X 的值，Context 的编码如图 **??**所示. 代码见函数 *ZeroCoding*.

D_0	V_0	D_1
H_0	X	H_1
D_2	V_1	D_3

表 4.1　待编码数据的 8 个相邻数据

LL and LH subbands (vertical high-pass)			HL subband (horizontal high-pass)			HH subband (diagonally high-pass)		Context label[a]
ΣH_i	ΣV_i	ΣD_i	ΣH_i	ΣV_i	ΣD_i	$\Sigma(H_i+V_i)$	ΣD_i	
2	x[b]	x	x	2	x	x	≥3	8
1	≥1	x	≥1	1	x	≥1	2	7
1	0	≥1	0	1	≥1	0	2	6
1	0	0	0	1	0	≥2	1	5
0	2	x	2	0	x	1	1	4
0	1	x	1	0	x	0	1	3
0	0	≥2	0	0	≥2	≥2	0	2
0	0	1	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0

图 4.1 零编码 (zero coding) 上下文 (context) 编码方式

符号编码 (Sign Coding)

该编码根据待编码的数据比特上、下、左、右的 4 个相邻数据重要性情况以及待编码数据的符号位编码，返回符号编码（XORbit）和 Context. 具体编码规则如表 **??**所示. 其中“1”表示在垂直或者水平方向的相邻两个数据都为重要并且符号都为正；或者只有一个是重要的情况. “0”表示两个方向的相邻数据中两个数据都不重要或者都为重要但是具有不同的符号. -1 表示的情况和 1 的情况相反. 代码见函数 *SignCoding*.

Horizontal contribution	Vertical contribution	Context label	XORbit
1	1	13	0
1	0	12	0
1	-1	11	0
0	1	10	0
0	0	9	0
0	-1	10	1
-1	1	11	1
-1	0	12	1
-1	-1	13	1

表 4.2 符号编码 (XORbit) 和上下文 (context) 编码方式

幅度精炼编码 (Magnitude Refinement Coding)

根据该数据位是否被第一次精炼，以及 $\sum H_i + \sum V_i + \sum D_i$ 的值进行编码，返回 D 和 Context．D 为 X 的值，Context 的编码规则如表 **??**所示．代码见函数 *MagnitudeRefinementCoding*．

是否第一次幅度精炼编码	$\sum H_i + \sum V_i + \sum D_i$	Context
否	x	16
是	≥ 1	15
是	0	14

表 4.3　幅度精炼编码 (Magnitude Refinement Coding) 上下文 (context) 编码方式

游程编码 (Run Length Coding)

仅当一个编码列 (4 个比特数据) 的目前状态都为不重要，且相邻元素也都为不重要时，开始进行游程编码处理．这时，如果一列中的 4 个数据也为不重要数据，则统一编码为一个上下文编码 CX=17 和编码数据 D=0．如果 4 个数据中至少有一个变为重要，则首先将其表示一个上下文 CX=17 和编码数据 D=1．然后，编码 4 个数据中的第一个重要数据的位置信息 (00 11) 作为 D，并编码上下文为 CX=18．此后，编码第一个重要数据的符号位，之后数据的编码按照零编码算法进行．具体编码规则如表 **??**所示．代码见函数 *RunLengthCoding*．

4 个比特数据	D	Context
(0,0,0,0)	[0]	[17]
(1,x,x,x)	[1,0,0] + 符号编码 + 零编码	[17,18,18]+ 符号编码 + 零编码
(0,1,x,x)	[1,0,1] + 符号编码 + 零编码	[17,18,18]+ 符号编码 + 零编码
(0,0,1,x)	[1,1,0] + 符号编码 + 零编码	[17,18,18]+ 符号编码 + 零编码
(0,0,0,1)	[1,1,1] + 符号编码	[17,18,18]+ 符号编码

表 4.4　游程编码 (Run Length Coding)D 和上下文 (context) 编码方式

4.2　分流通道

重要性传播编码通道

在此通道中，编码位平面中的目前状态为“无效态”，但有很大概率成为“有效态”的比特数据．只要四周的 8 个比特数据有一个是重要的，就将被进行零编码 (Zero Coding)．在此基础上，如果该位的比特数为 1，再进行符号编码．流程图如图 **??**所示．代码见函数 *SignificancePropagationPass*．

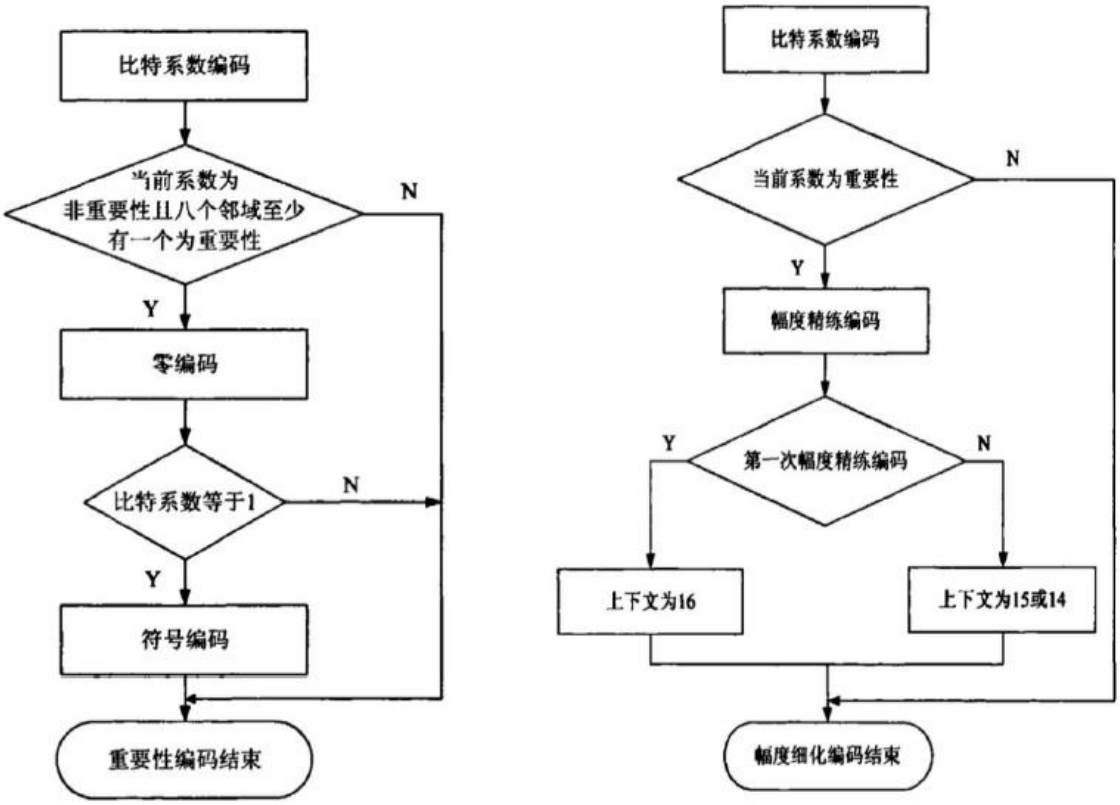


图 4.2 重要性传播编码通道（左）及幅度精炼编码通道（右）编码流程图

幅度精炼编码通道

在此通道中，编码那些在先前位平面已经被判定为有效态的数据，用“幅度精炼编码”编码．流程图如图 **??**所示．代码见函数 *MagnitudeRefinementPass*．

清除编码通道

在此通道中，编码位平面中所有还未编码的数据．当一列 4 个比特都在此通道中被编码，且这 4 个比特都没有重要的相邻数据，则对 4 个比特采用游程编码 (RLC)，否则分别对每个比特采用零编码 (Zero Coding)，如数据为 1，再进行符号编码 (Sign Coding)．流程图如图 **??**所示．代码见函数 *CLeanUpPass*．

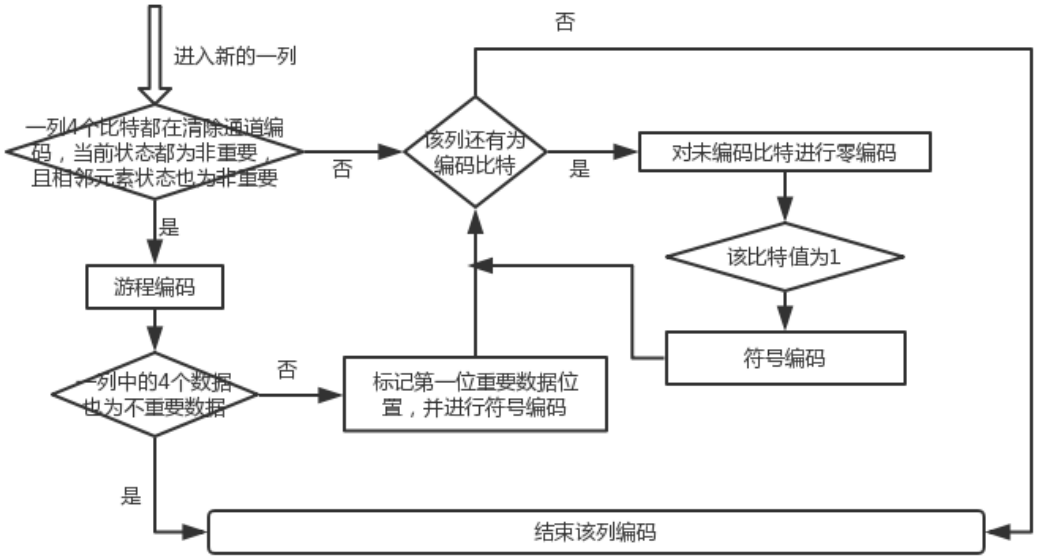


图 4.3　清除编码通道编码流程图

4.3　编码流程

区段分块

将上一步量化后的频块进一步切成 32×32 的区段，作为内嵌区段编码的对象，各区段之间独立运算．如频块不能被 32 整除，则用 0 补足，但是补零的同时会增加编码的冗余.

位元层及符号矩阵

对把编码区段内的数据取绝对值，依照位元深度，从高位元（MSB）到低位元（LSB）分成 8 个位元层．此外，由数据的正负组成符号矩阵，其中 0 表示正数，1 表示负数.

初始化上下文、重要性等矩阵

在对整个区段编码的过程中，需要用到的矩阵及其含义如表 **??**所示.

矩阵（向量）名称	大小	含义
重要性矩阵 (S1)	32*32	表明当前状态是否为重要（用于零编码及游程编码）
精炼标记矩阵 (S2)	32*32	表明该元素是否被精炼（用于精炼编码）
编码标记矩阵 (S3)	32*32	表明该元素是否被编码（用于精炼通道及清除编码通道）
符号矩阵 (signs)	32*32	表明该元素的符号（用于符号编码）
位元层 (bitplanes)	X*32*32	小波变化量化后的 2 进制位元表示
D	N*1	编码输出矩阵
上下文 (CX)	N*1	编码输出上下文矩阵

表 4.5　编码中使用的矩阵说明

逐位元层编码

由高位至低位，对各位元层编码．对于各位元层，依次按照重要性传播编码通道，幅度精炼编码通道，以及清除编码通道的顺序编码．在编码各通道时，按照如图 **??**所示的方式扫描．扫描过程从位平面左上角的数据开始，以 4 个数据为一组，连续扫描第一列的第一组 4 个数据后，然后转向扫描第二列的第一组 4 个数据，如此一直扫描到最后一列的第一组 4 个数据．然后，转向扫描第一列的第二组 4 个数据，一直到最后一列的第二组 4 个数据．按照这样的顺序依次扫描整个位平面．扫描过程中矩阵的更新情况如表 **??**所示．代码见函数 *codeBlockfun*.

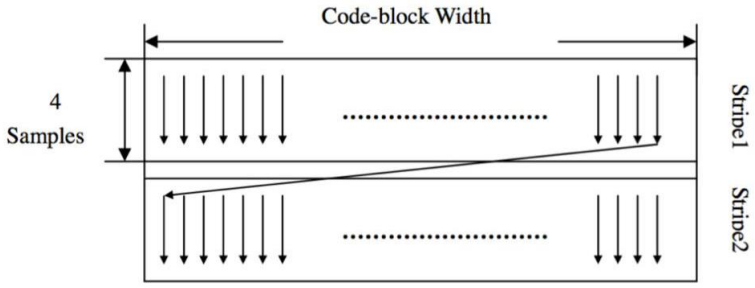


图 4.4　位平面扫描顺序示意图

矩阵（向量）名称	更新时间
重要性矩阵 (S1)	重要性传播编码通道及清除编码通道中更新
精炼标记矩阵 (S2)	精炼通道中更新
编码标记矩阵 (S3)	重要性传播编码通道及精炼通道中更新，结束一层位平面编码清零
D	各通道都更新
上下文 (CX)	各通道都更新

表 4.6　扫描过程中矩阵的更新情况说明

5　MQ 算术编解码器

5.1　编码器结构

JPEG2000 所用的 MQ 编码器结构如图**??**所示．输入由待编码位 D 和上下文矢量 CX 构成，输出为压缩之后生成的码流.

MQ 编码器是一种自适应二进制算术编码器．其中，自适应算术编码是指编码系统根据已经传输和编码的信息串，调整用来划分区间的当前符号概率估计．对于同一待编码位 D，其上下文 CX 不同，则对应的符号概率并不一定相同．因此，必须读取 CX 当前状态的符号概率来确定消息的符号概率，同时决定是否要转移到下一状态．符号概率的状态转移表由标准提供，共 47 种状态.

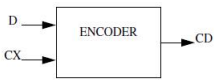


图 5.1　MQ 编码器结构图

5.2　递归区间划分

算术编码器按符号概率将符号分为大概率符号 MPS 和小概率符号 LPS. 如图 **??**所示，LPS 区间在 MPS 区间之前. 其中，A 表示概率间隔长度，C 表示概率间隔下限，则 [C,C+A] 即为编码区间. 如果 LPS 当前的概率估计值是 Qe，编码当前符号后子区间的变化如下：

如果输入符号是 MPS，$A = A * (1 - Q_e), C = C + A * Q_e$

如果输入符号是 LPS，$A = A * Q_e, C$ 不变

由于乘法运算比较费时，为简化计算，可将区间 A 保持在十进制 [0.75,1.5] 的范围内，此时 A 和 C 可近似为：

如果输入符号是 MPS，$A = A - Q_e, C = C + Q_e$

如果输入符号是 LPS，$A = Q_e, C$ 不变

为了将 A 保持在 [0.75,1.5] 的范围内，当整数值降到小于 0.75 时，需通过重归一化把它加倍. 重归一化发生时，调用概率估计模型，为当前正被编码的上下文确定一个新的概率估计. 若将 A 限制在十进制范围 0.75-1.5 之间，概率区间的划分可以使用简单的算术近似方法.

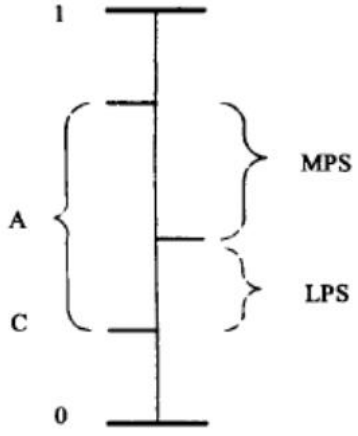


图 5.2　MQ 编码器编码区间示意图

5.3　编码流程

首先对编码器进行初始化 INTENC，然后读入上下文 CX 和待编字节 D 开始编码. 编码时判断 D 是大概率符号还是小概率符号. 如为大概率符号，进行 CODEMPS 编码；如为小概率符号，进行 CODELPS 编码. CODEMPS 和 CODELPS 的编码流程如图**??**和图**??**所示. 代码见函数 *entropy_coding*.

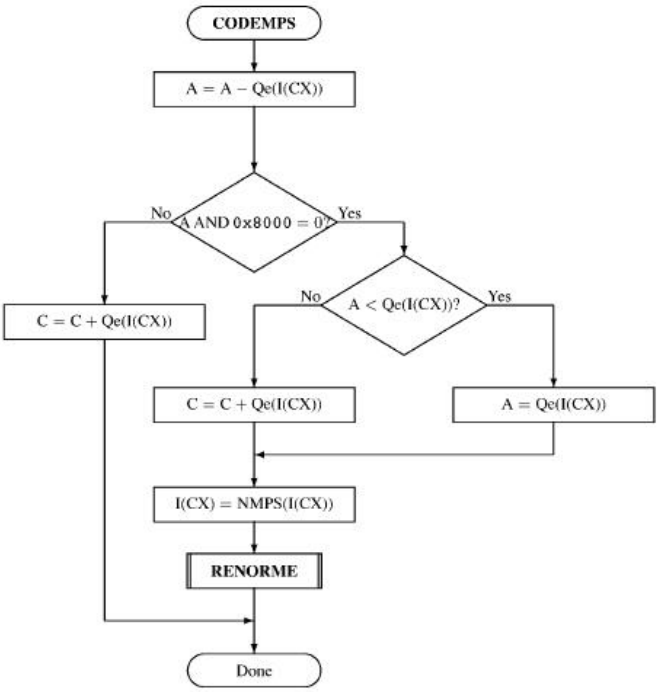


图 5.3 CODEMPS 流程图

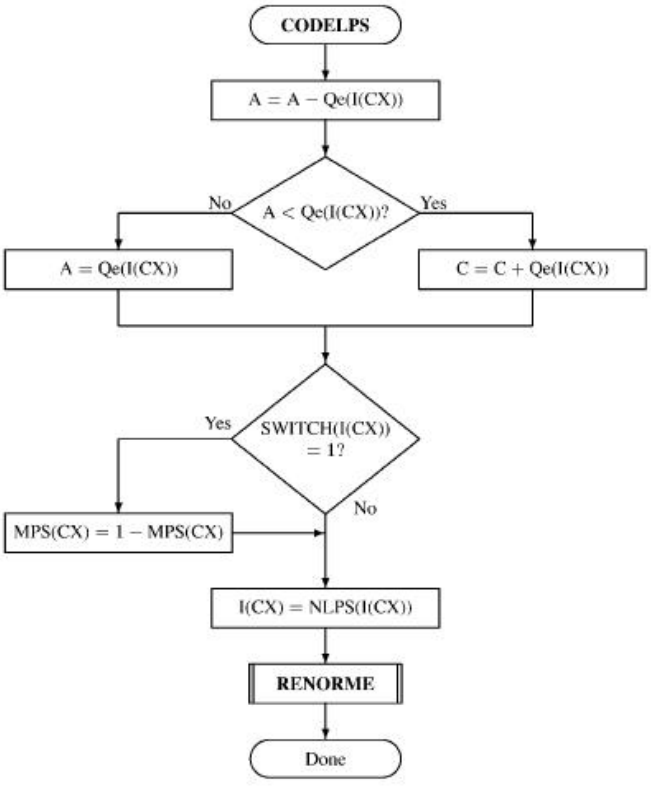


图 5.4 CODELPS 流程图

5.4 解码流程

解码器根据 CX 和 CD，一次解码一个二进制数据（如图**??**所示）. 同编码时一样，在解码工程中，同步调整用来划分区间的当前符号概率估计. 代码见函数 *entropy_decoding*.

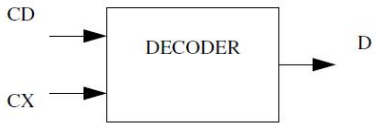


图 5.5 MQ 解码器结构图

6 生成数据流

将生成的 stream 和 Context 储存为二进制文件，每一个整数用 4 个字节保存. context 的范围为 0-18，stream 的范围为 0-255. 由于分块后 tile 的大小为 1024*1024，故小波变换后各频带“图像”的大小小于 1024. 因此，可选用大于 1024 的整数作为分隔符. 不同 tile 之间用 2051 分隔，不同频带之间用 2050 分隔，不同 block(64*64) 之间用 2049 分隔，CX 和 stream 之间用 2048 分隔. 代码见函数 *image_entropy, tile_entropy, block_entropy*.

由于小波变换后的各频带” 图像” 大小并不是 64 的整数，在进行内嵌区段编码前需要补零，才能将图像分割为 m*n 个 64*64 的 block. 因此，在数据流中还需要保存“图像”大小. 这样，解码时将补零后的图像恢复成原先的大小.

7 结果

由于程序在图片过大、颜色位数过大时运行很慢（$1080 \times 1920 \times 3$ 的 24 位真彩色图片要运行 30 分钟），所以在 demo 中我们以 $512 \times 512 \times 3$ 的 8 位图片 Lena 为例进行了测试.

图 7.1 Lena.

7.1 小波变换结果分析

选取第一个 tile 中的 y 分量分解得到的三级小波系数（这里以 5/3 LG 滤波器为例），将其按照**??**所示的二维三级小波变换的区域划分进行排列. 从图**??**中可以看到，第一级小波分解得到的 LH,HL 和 HH 分量中仍含有很多的图片纹理信息，而对 LL 分量进行进一步分解之后得到的二级和三级小波系数中纹理逐渐消失，说明信息得到了充分的分解. 对于 9/7 DB 滤波器也有类似的结果. 运行代码的时候将可选参数设为 $debug = True$ 即可显示出小波系数可视化的图片.

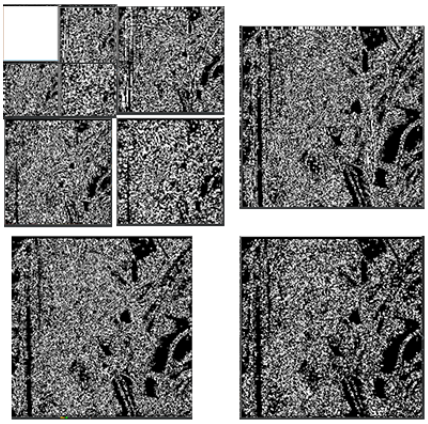


图 7.2 小波系数的可视化.

7.2 压缩前后比较

由生成压缩图像的大小来看，压缩效果并不理想，原因有二：

1. 需要存储的数据，绝大多数（除” 图像” 大小以外）都是在 0-255 的范围内，使用 4 个字节存储会有冗余.

2. 在解码时需要使用 Context 来估计 0,1 的概率分布，但是 Context 本身并不能直接反映图像内容. 而且在一个 block 中，Context 所占的字节数远大于 Stream 的字节数（我们也存储了只含有 stream 的二进制文件，比较两者的文件大小即可发现）. 因此，**如何处理 Context 还需要进一步探究**.

参考文献

[1] jpeg2000. Jordan Van Duyne. https://github.com/jovanduy/jpeg2000.

[2] Taubman D , Marcellin M , 陶布曼, et al. JPEG2000 图像压缩基础、标准和实践 [M]. 电子工业出版社, 2004.

[3] CSDN: 压缩算法——JPEG2000 编解码原理. 佚名. https://blog.csdn.net/ytang_/article/details/76571635.

[4] 百度文库:Daubechies 97 和 Le Gall 53 小波分解与合成. 佚名. https://wenku.baidu.com/view/84fc5c7caaea998fcc220e7b.

[5] 知乎: 能不能通俗的讲解下傅立叶分析和小波分析之间的关系? . 佚名. https://www.zhihu.com/question/22864189/answer/40772083.

[6] pywavelets 官方文档. 佚名. https://pywavelets.readthedocs.io/en/latest/index.html.