

# 图像压缩编码

Ji Zhang

zhangji@glassix.com

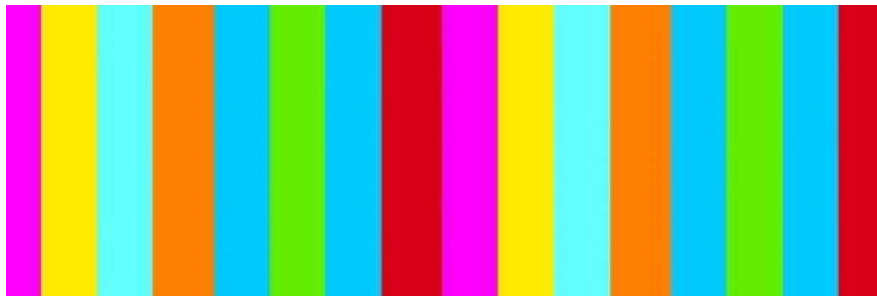
2018/8/31

# 编码冗余

对信息序列 aaaabbbbc 使用如下两种编码方式：

- ① 2bit 定长编码：a(00)、b(01)、c(10)，则平均码长为 2。
- ② 可变长度编码：a(0)、b(1)、c(00)，则平均码长为  $(4 \times 1 + 3 \times 1 + 1 \times 2) / 8 = 1.125$ 。

# 空间冗余



# 时间冗余



## 霍夫曼编码

Original source		Source reduction				
Symbol	Probability	1	2	3	4	
$a_2$	0.4	0.4	0.4	0.4	0.6 0.4	
$a_6$	0.3	0.3	0.3	0.3		
$a_1$	0.1	0.1	0.2	0.3		
$a_4$	0.1	0.1				
$a_3$	0.06	0.1	0.1			
$a_5$	0.04					

Original source			Source reduction							
Symbol	Probability	Code	1	2		3		4		
$a_2$	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
$a_6$	0.3	00	0.3	00	0.3	00	0.3	00		0.4
$a_1$	0.1	011	0.1	011	0.2	010	0.3	01		
$a_4$	0.1	0100	0.1	0100	0.1	011				
$a_3$	0.06	01010	0.1	0101						
$a_5$	0.04	01011								

# Golomb 编码

对给定正整数  $m$  和  $n$ , 表示为  $G_m(n)$  的  $n$  关于  $m$  的 Golomb 编码是商  $\text{floor}(\frac{n}{m})$  的一元编码和  $n \bmod m$  的二进制表示的一个合并。

步骤 1 形成商  $\lfloor n/m \rfloor$  的一元编码 (整数  $q$  的一元编码定义为  $q$  个 1 紧跟着一个 0)。

步骤 2 令  $k = \lceil \log_2 m \rceil, c = 2^k - m, r = n \bmod m$ , 并计算截短的余数  $r'$ , 例如, 使其满足

$$r' = \begin{cases} r \text{ 截短至 } k-1 \text{ 比特,} & 0 \leq r < c \\ r+c \text{ 截短至 } k \text{ 比特,} & \text{其他} \end{cases}$$

步骤 3 连接步骤 1 和步骤 2 的结果。

计算  $G_4(9)$ :

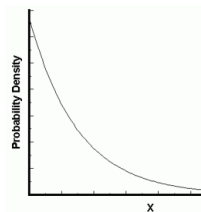
Step1:  $\text{floor}(\frac{9}{4}) = \text{floor}(2.25) = 2$ , 商的一元编码为 110

Step2:  $k = \text{ceil}(\log_2 4) = 2, c = 2^2 - 4 = 0, r = 9 \bmod 4 = 1$ , 截短后的  $r'$  为 01

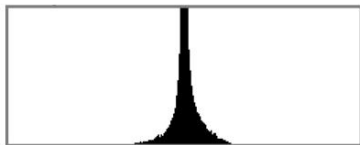
Step3: 连接后的  $G_4(9) = 11001$

# Golomb 编码

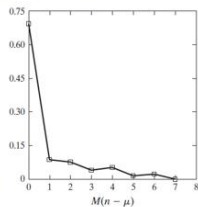
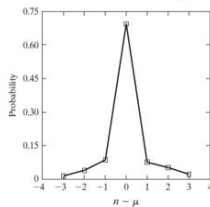
$n$	$G_1(n)$	$G_2(n)$	$G_4(n)$	$G_{\text{exp}}^0(n)$
0	0	00	000	0
1	10	01	001	100
2	110	100	010	101
3	1110	101	011	11000
4	11110	1100	1000	11001
5	111110	1101	1001	11010
6	1111110	11100	1010	11011
7	11111110	11101	1011	1110000
8	111111110	111100	11000	1110001
9	1111111110	111101	11001	1110010



# Golomb 编码



$$M(n) = \begin{cases} 2n & n \geq 0 \\ 2|n| - 1 & n < 0 \end{cases}$$





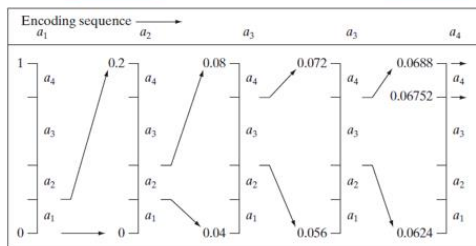
# 算术编码

算术编码为整个信源符号序列分配一个单一的算术码字,采用子区间划分的方法, 计算公式:

$\text{range} = \text{high} - \text{low}$ ;  $\text{high} = \text{low} + \text{range} * \text{probHigh}$ ;  $\text{low} = \text{low} + \text{range} * \text{probLow}$ .

Source Symbol	Probability	Initial Subinterval
$a_1$	0.2	[0.0, 0.2)
$a_2$	0.2	[0.2, 0.4)
$a_3$	0.4	[0.4, 0.8)
$a_4$	0.2	[0.8, 1.0)

- ①  $a_1$ :  $\text{range} = 1.0 - 0.0 = 1.0$ ;  
 $\text{high} = 0.0 + 1.0 * 0.2 = 0.2$ ;  
 $\text{low} = 0.0 + 1.0 * 0.0 = 0.0$ .
- ②  $a_2$ :  $\text{range} = 0.2 - 0.0 = 0.2$ ;  
 $\text{high} = 0.0 + 0.2 * 0.4 = 0.08$ ;  
 $\text{low} = 0.0 + 0.2 * 0.2 = 0.04$ .



$a_1 a_2 a_3 a_3 a_4 \Rightarrow [0.06752, 0.0688)$

# 算术编码

基于自适应二元算术编码: CABAC(Context-based Adaptive Binary Arithmetic Coding)

① Initial range: [0.0, 1.0)  $\Rightarrow$  Initial range: [0, 0xFFFFFFFF)

②

Source Symbol	Probability	Initial Subinterval
$a_1$	0.2	[0.0, 0.2)
$a_2$	0.2	[0.2, 0.4)
$a_3$	0.4	[0.4, 0.8)
$a_4$	0.2	[0.8, 1.0)

$\Rightarrow$

Source Symbol	Probability	Initial Subinterval
0	$P_0$	[ <u>probLow</u> * 0xFFFFFFFF, <u>probHigh</u> * 0xFFFFFFFF)
1	$1 - P_0$	[ <u>probLow</u> * 0xFFFFFFFF, <u>probHigh</u> * 0xFFFFFFFF)

③ 自适应更新概率: 
$$p^{(t+1)}_{LPS} = \begin{cases} \alpha \cdot p^t_{LPS} & \text{if an MPS occurs} \\ \alpha \cdot p^t_{LPS} + (1 - \alpha) & \text{if an LPS occurs} \end{cases}$$

注: MPS(Most Probability Symbol), LPS(Low Probability Symbol), 分别代表出现概率大小的符号;  
 $\alpha = 0.95$ 。

# 行程编码

行程编码RLE (Run Length Encoding) 算法把数据看成一个存在连续重复数据块的线性序列。采用的压缩策略是用一个字节表示数据块重复的次数，然后存储对应的数据字节本身。

AAAAABCCCC => 6A1B4C

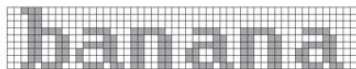
ABC => 1A1B1C

像素变化次数较多时可能导致数据扩展，因此行程编码较适合二值图像：

0001111110000011000111	=>	3, 6, 5, 2, 3, 3	
1111100000111111100000		0, 5, 5, 7, 5	默认每行起始字符为“0”
0001100011111000000011		3, 2, 3, 5, 7, 2	

# 基于符号的编码

一幅图像表示为多幅频繁发生的子图像（**符号**）的一个集合。每一个**符号**都存储在符号字典中，且该图像以一个三元组 $\{(x_1, y_1, t_1), (x_2, y_2, t_2), \dots\}$ 的集合来编码，其中 $(x_i, y_i)$ 表示图像中符号的位置， $t_i$ 是该符号在字典中的地址。



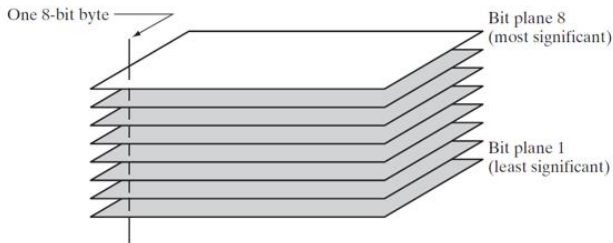
Token	Symbol
0	
1	
2	

Triplet
(0, 2, 0)
(3, 10, 1)
(3, 18, 2)
(3, 26, 1)
(3, 34, 2)
(3, 42, 1)

# 比特平面编码

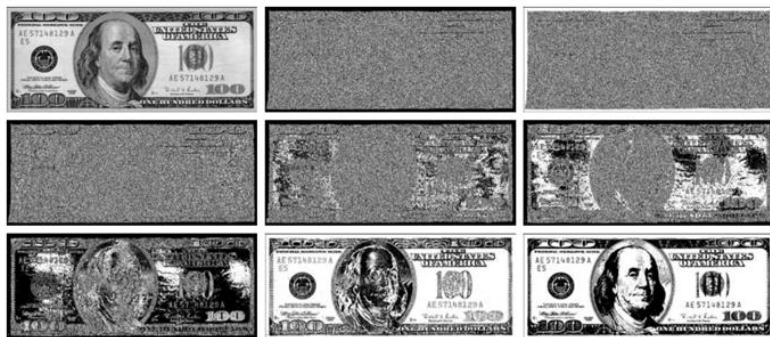
将一幅多级（彩色）图像分解为一系列二值图像，然后使用某种二值压缩方法（行程编码）来压缩每幅二值图像。

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \dots + a_12^1 + a_02^0$$



# 比特平面编码

0阶平面对灰度值贡献最大为 $2^0=1$ , 1阶平面对灰度值贡献最大为 $2^1=2$ , 2阶平面对灰度值贡献最大为 $2^2=4$ , 直接舍弃这3个平面可减少 $3/8=37.5\%$ 的数据量。



# 块变换编码

块变换编码将图像分成大小相等（如 $8 \times 8$ ）且不重叠的小块，然后用可逆线性变换把每个小块映射为变换系数的集合。变换后的大多数系数都有较小的值，因此可以参照降维原理（主成分分析 PCA、奇异值分解 SVD），取一定比例的最大幅值，在保证图像质量的同时减少数据量。

$$\text{离散变换: } T(u, v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x, y) r(x, y, u, v)$$

$$\text{离散逆变换: } g(x, y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) s(x, y, u, v)$$

$r(x, y, u, v)$  和  $s(x, y, u, v)$  分别称为正变换核和逆变换核

# 块变换编码

根据所使用变换核的不同，主要有3种变换形式：

① Walsh-Hadamard变换(WHT):  $r(x, y, u, v) = s(x, y, u, v) = \frac{1}{n} (-1)^{\sum_{i=0}^{m-1} [b_i(x)p_i(u) + b_i(y)p_i(v)]}$

② 离散傅里叶变换(DFT):

$$r(x, y, u, v) = e^{-j2\pi(ux+vy)/n}$$

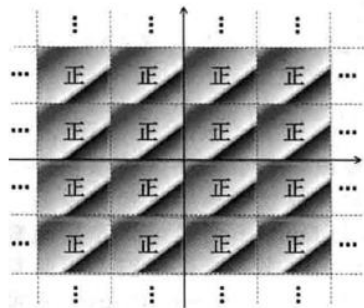
$$s(x, y, u, v) = \frac{1}{n^2} e^{j2\pi(ux+vy)/n}$$

③ 离散余弦变换(DCT):

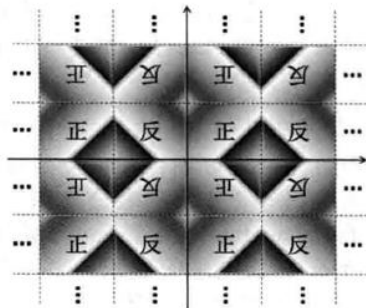
$$r(x, y, u, v) = s(x, y, u, v) = \alpha(u)\alpha(v) \cos\left[\frac{(2x+1)u\pi}{2n}\right] \cos\left[\frac{(2y+1)v\pi}{2n}\right]$$



# 块变换编码



(a) DFT所对应的周期延拓



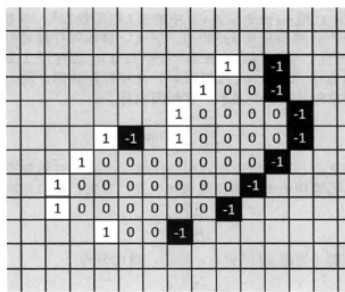
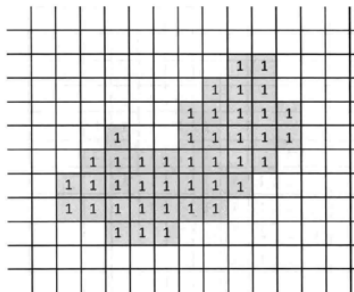
(b) DCT所对应的周期延拓

$$e^{j\theta} = \cos \theta + j \sin \theta$$

$$\int_{-\infty}^{+\infty} (a \cos x + b \sin x) dx = \int_{-\infty}^{+\infty} (a \cos x) dx$$

# 预测编码

通过消除紧邻像素在空间和时间上的冗余来实现，仅对每个像素中的新信息进行提取和编码。

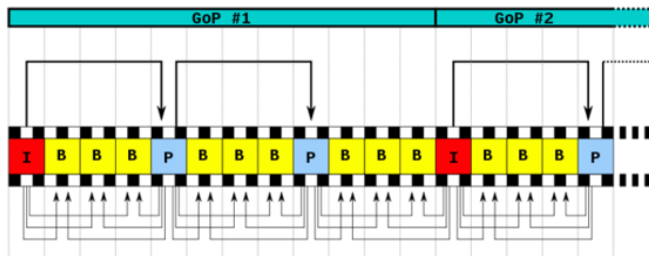


$$\text{差分编码: } e(x, y) = f(x, y) - f(x, y - 1)$$

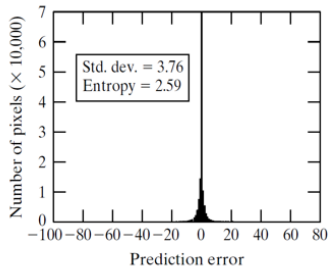
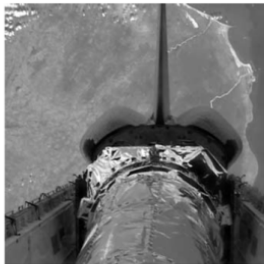
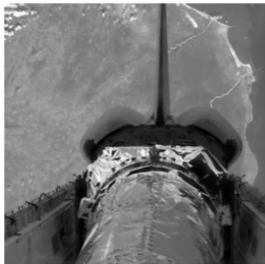
# 预测编码

- **I帧**：关键帧，完整编码的帧，可独立重建图像。
- **P帧**：前向参考帧，只包含与前一帧差异部分编码的帧，需结合前一帧才能重建图像。
- **B帧**：双向参考帧，既参考前面的帧，又参考后面的帧，需结合前后帧才能重建图像。

图像序列GOP(Group Of Pictures)：一个序列就是一段内容差异不太大的图像编码后生成的一串数据流，两个I帧之间是一个图像序列。



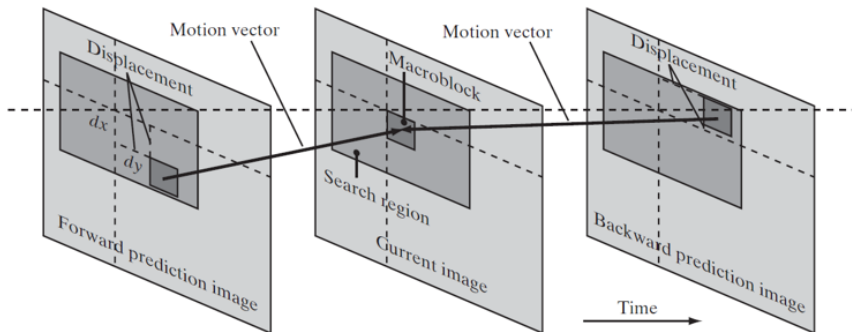
# 预测编码



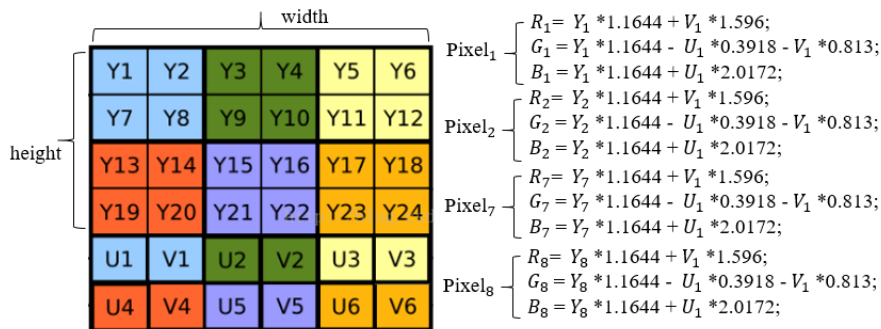
预测残差:  $e(x, y, t) = f(x, y, t) - f(x, y, t - 1)$

# 预测编码

运动补偿：在预测和差分处理期间，跟踪目标运动并对其进行补偿。



## 颜色空间转换



数据量:  $height * width + 1/4 * height * width + 1/4 * height * width = 3/2 * height * width$ , 减少50%

# 压缩解码流程

