

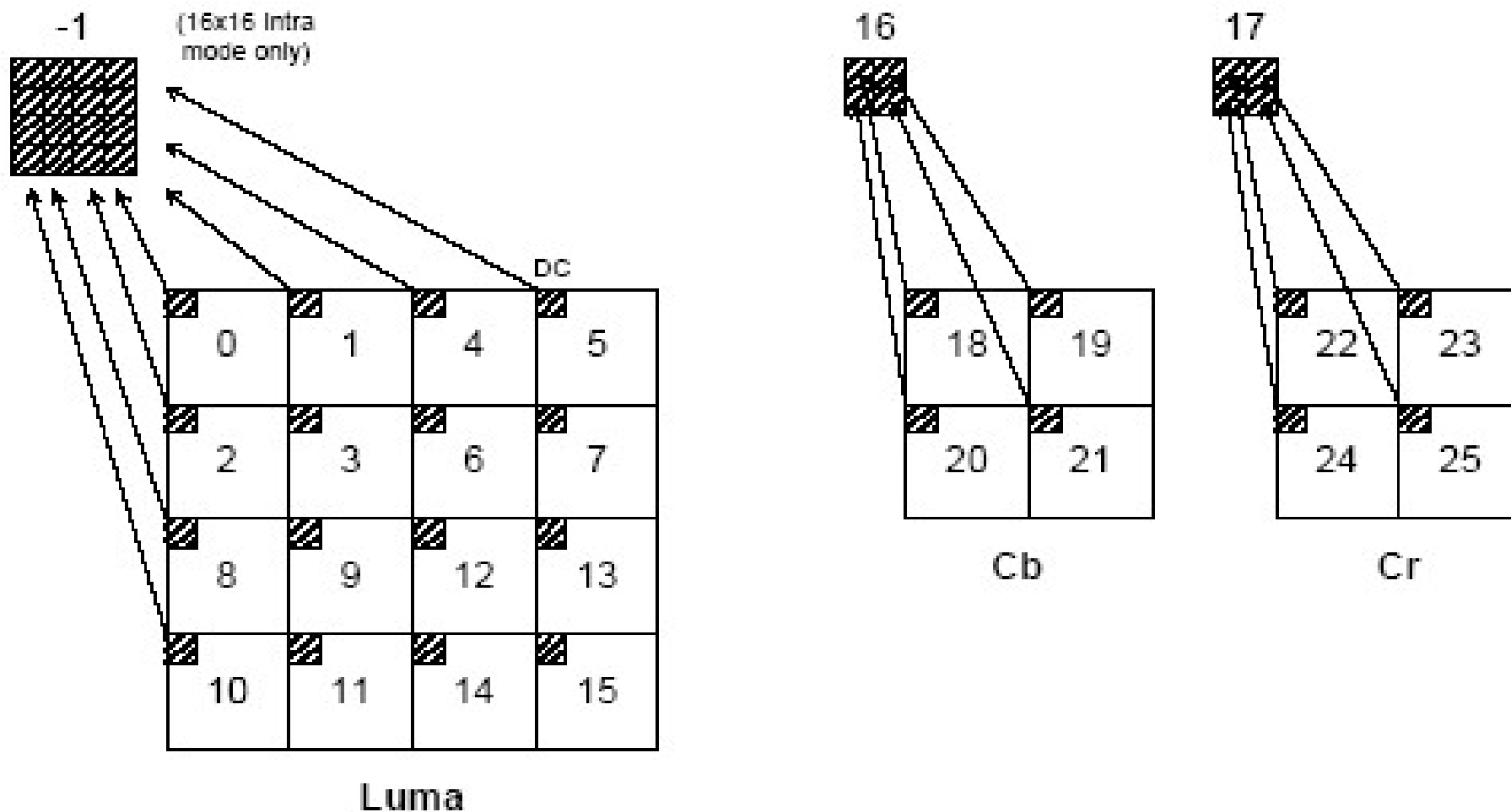
第五讲、H.264压缩编码标准

Communication University of China

4、整数变换

- 对每个宏块的预测误差都将进行变换，量化和编码。
H.264的“**Baseline**”型（**profile**）根据所压缩的数据类型不同而采用三种不同的变换方式。
- 适用于帧内预测宏块 4×4 亮度**DC**系数块的变换；
- 适用于任何宏块色度 2×2 **DC**系数块的变换；
- 适用于其他 4×4 残差数据块的变换；
- 如果需要，还可以选择与运动补偿块大小（ 4×8 、 8×4 、 8×8 、 16×8 ）相对应的变换。

宏块扫描和传输顺序



(1) 4×4 残差变换

- 变换适用于运动补偿或帧内预测后的 4×4 残差数据块（0~15和18~25）。虽然该变换的基础是DCT变换但二者之间有根本的差别：
- 1)该变换是一种整数变换，所有的运算都是整数运算并且没有精度损失。
- 2)H.264标准中对该变换的反变换有详细的说明，如果完全按照说明正确执行，编解码器之间不会出现误匹配。
- 3)该变换的核心部分不需要乘法，仅仅需要加法和移位运算。



N=4的DCT变换核函数展开

核函数 $a(u, x) = C(u) \cos \frac{(2x+1)u\pi}{2N} \quad x=0,1,\dots,N-1$

$$C(u) = \begin{cases} \sqrt{\frac{1}{N}} & k=0 \\ \sqrt{\frac{2}{N}} & k=1,2,\dots,N-1 \end{cases}$$

$$\frac{1}{\sqrt{2}} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \cos \frac{\pi}{8} & \cos \frac{3\pi}{8} & \cos \frac{5\pi}{8} & \cos \frac{7\pi}{8} \\ \cos \frac{2\pi}{8} & \cos \frac{6\pi}{8} & \cos \frac{10\pi}{8} & \cos \frac{14\pi}{8} \\ \cos \frac{3\pi}{8} & \cos \frac{9\pi}{8} & \cos \frac{15\pi}{8} & \cos \frac{21\pi}{8} \end{pmatrix}$$

N=4的DCT变换核函数展开

核函数 $a(u, x) = C(u) \cos \frac{(2x+1)u\pi}{2N} \quad x=0,1,\dots,N-1$

$$\frac{1}{\sqrt{2}} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \cos \frac{\pi}{8} & \cos \frac{3\pi}{8} & -\cos \frac{3\pi}{8} & -\cos \frac{\pi}{8} \\ \cos \frac{\pi}{4} & -\cos \frac{\pi}{4} & -\cos \frac{\pi}{4} & \cos \frac{\pi}{4} \\ \cos \frac{3\pi}{8} & -\cos \frac{\pi}{8} & \cos \frac{\pi}{8} & -\cos \frac{3\pi}{8} \end{pmatrix}$$

N=4的DCT变换核函数展开

核函数 $a(u, x) = C(u) \sum \cos \frac{(2x+1)u\pi}{2N} \quad x=0,1,\dots,N-1$

$$\frac{1}{\sqrt{2}} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \cos \frac{\pi}{8} & \cos \frac{3\pi}{8} & -\cos \frac{3\pi}{8} & -\cos \frac{\pi}{8} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \cos \frac{3\pi}{8} & -\cos \frac{\pi}{8} & \cos \frac{\pi}{8} & -\cos \frac{3\pi}{8} \end{pmatrix}$$

N=4的DCT变换核函数展开

核函数 $a(u, x) = C(u) \cos \frac{(2x+1)u\pi}{2N} \quad x=0,1,\dots,N-1$

令 $a = \frac{1}{2}, b = \frac{1}{\sqrt{2}} \cos \frac{\pi}{8}, c = \frac{1}{\sqrt{2}} \cos \frac{3\pi}{8},$

$$A = \begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{pmatrix}$$

DCT变换:

$$W = A \cdot X \cdot A^T$$

N=4的DCT变换核函数展开

核函数 $a(u, x) = C(u) \cos \frac{(2x+1)u\pi}{2N} \quad x=0,1,\dots,N-1$

令 $a = \frac{1}{2}, b = \frac{1}{\sqrt{2}} \cos \frac{\pi}{8}, c = \frac{1}{\sqrt{2}} \cos \frac{3\pi}{8},$

$$\begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{pmatrix} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ b/c & 1 & -1 & -b/c \\ 1 & -1 & -1 & 1 \\ 1 & -b/c & b/c & -1 \end{pmatrix}$$

N=4的DCT变换核函数展开

核函数 $a(u, x) = C(u) \cos \frac{(2x+1)u\pi}{2N} \quad x=0,1,\dots,N-1$

$$b/c = \frac{1}{\sqrt{2}} \cos \frac{\pi}{8} / \frac{1}{\sqrt{2}} \cos \frac{3\pi}{8} = 2.41$$

$$\begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{pmatrix} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix}$$

N=4的DCT变换核函数展开

$$A = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix}$$

$$A^T = \begin{pmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{pmatrix} = \begin{pmatrix} 1 & b/c & 1 & 1 \\ 1 & 1 & -1 & -b/c \\ 1 & -1 & -1 & b/c \\ 1 & -b/c & 1 & -1 \end{pmatrix} \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix}$$

N=4的DCT变换核函数展开

$$\begin{aligned}
 A^T &= \begin{pmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{pmatrix} = \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix} \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix} \\
 &\quad C \cdot X \cdot C^T \\
 &= \underbrace{\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix}}_A \underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix}}_{A^T} \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix}
 \end{aligned}$$

N=4的DCT变换核函数展开

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} X \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix} \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} X \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix} [1] \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix}$$

N=4的DCT变换核函数展开

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} X \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix} \Rightarrow \begin{pmatrix} a & a & a & a \\ c & c & c & c \\ a & a & a & a \\ c & c & c & c \end{pmatrix} \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & c \end{pmatrix} = \begin{pmatrix} a^2 & ac & a^2 & ac \\ ac & c^2 & ac & c^2 \\ a^2 & ac & a^2 & ac \\ ac & c^2 & ac & c^2 \end{pmatrix}$$

N=4的DCT变换核函数展开

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} X \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} a^2 & ac & a^2 & ac \\ ac & c^2 & ac & c^2 \\ a^2 & ac & a^2 & ac \\ ac & c^2 & ac & c^2 \end{pmatrix}$$

$b/c=2, \quad c=b/2 \quad C \cdot X \cdot C^T$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} X \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{pmatrix}$$

整数变换矩阵

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \quad C^T = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

$$W = CXC^T$$

- (2) **Hadamard**变换
- 4×4 , 针对亮度块的**直流DC**系数;
- 2×2 , 针对色度块的**直流DC**系数;

$$c = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \quad \longrightarrow \quad h = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

$$h' = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$



5、H.264量化

- 通过量化，在不降低图像视觉效果的前提下,减少图像编码长度。

$$FQ = round\left(\frac{Y_{sample}}{Q_{step}}\right)$$

H264量化步长特点

- H.264标准支持52个量化步长，对应于不同的量化参数（QP）如表所示，**QP值每增加6，Qstep值增加一倍**。量化步长取值范围很广，这就为编码中兼顾比特率和编码质量提供了足够多的灵活度和准确度。

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	...
Qstep	0.625	0.6875	0.8125	0.875	1	1.125	1.25	1.375	1.625	1.75	2	2.25	2.5	...
QP	...	18	...	24	...	30	...	36	...	42	...	48	...	51
Qstep		5		10		20		40		80		160		224

$$Z_{ij} = \text{round}\left(W_{ij} \frac{PF}{Q_{step}}\right) \quad MF = \frac{PF}{Q_{step}} 2^{qbits}$$

$$Z_{ij} = \text{round}\left(W_{ij} \frac{MF}{2^{qbits}}\right)$$

PF=

$$\begin{pmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{pmatrix}$$

MF值:

QP	a^2	$b^2/4$	$ab/2$
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

6、去块效应滤波

- **H.264/AVC**定义了一个自适应循环滤波器，滤波的强度通过几个语法元素控制。
- 滤波的基本思想是：如果块边沿的绝对差值相对比较大，出现块人工瑕疵的可能性就很大，因此需要进行相应处理。

Filter



Figure 6.32 Original frame (violin frame 2)



Figure 6.33 Reconstructed, $QP = 36$ (no filter)



Figure 6.34 Reconstructed, $QP = 36$ (with filter)



Figure 6.35 Reconstructed, $QP = 32$ (no filter)



Figure 6.36 Reconstructed, $QP = 32$ (with filter)

7、熵编码

- **H.264**标准提供的熵编码方案有：
- 通用变长码编码(**UVLC**)；
- 基于上下文的自适应变长码编码(**CAVLC**)；
(**Exponential Golomb Codes**)
- 基于上下文的自适应二进制算术编码(**CABAC**)。

(1)、Exp-Golomb编码

码字序号	码字
------	----

0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000

...

...

- 对所有的句法元素，除了量化系数外，使用单一无限扩展的码字表。
- 是有规则结构的可变长编码。

(2)、CAVLC编码

- 遵循变长编码的思路，即根据概率统计分布制定码表，通过信源符号与码字相对应实现编码。
- 通过根据已编码句法元素的情况，动态调整编码中使用的码表，从而取得极高的压缩比。

(2)、CAVLC编码

- 用于亮度和色度残差块变换系数的编码。利用了 4×4 块的一些特性：
- 预测变换量化后的块一般是稀疏的。
- 之字形扫描后的最高非零系数是+1/-1的序列。
- 相邻块的非零系数是相关的。
- 非零系数的幅度在重排数组的开始处比较高，在高频系数比较低。

Example1

-2 4 0 -1

3 0 0 0

-3 0 0 0

0 0 0 0

- 重排后数据是:
- -2, 4, 3, -3, 0, 0, -1, 0, ...
- 对非零系数的数目和绝对值为“1”的数目进行编码;
- 对各个T1的符号进行编码。T1的符号用1个比特编码0表示+, 1表示-。从最高频率的T1开始反向编码。
- 按逆向扫描的次序对余下的数值进行编码。
- 对最后一个系数之前的零的总数进行编码。
- 对零的游程进行编码, 说明零是如何分布的。

利用已编码符号和将要编码符号的 相关性

- 根据**相邻已编码块**使用的表格确定当前块非零系数数目和绝对值为**1**的系数数目编码所用的表格；
- 根据**已编码非零系数的幅度**决定当前非零系数幅度编码所使用的表格。

N	Table for coeff_token
0, 1	Num-VLC0
2, 3	Num-VLC1
4, 5, 6, 7	Num-VLC2
8 or above	FLC



Current VLC table	Threshold to increment table
VLC0	0
VLC1	3
VLC2	6
VLC3	12
VLC4	24
VLC5	48
VLC6	N/A (highest table)

Example1







TotalCoeffs = 5 (indexed from highest frequency [4] to lowest frequency [0])

TotalZeros = 2

T1s = 1

$nA=2, nB=1, nC = (nA + nB) \gg 1 = 1$

Encoding:

Element	Value	Code
coeff_token	TotalCoeffs=5, T1s=1	0000000110 
T1 sign (4)	-	1
Level (3)	Sent as -2 (see note 1) (use Level_VLC0)	0001 
Level (2)	3 (use Level_VLC1)	0010 
Level (1)	4 (use Level_VLC1)	00010
Level (0)	-2 (use Level_VLC2)	111 
TotalZeros	2	0011 
run_before(4)	ZerosLeft=2; run_before=2	00 
run_before(3..0)	0	No code required

The transmitted bitstream for this block is 000000011010001001000010111001100.

Example1

Note 2: After encoding level (3), the level_VLC table is incremented because the magnitude of this level is greater than the first threshold (which is 0). After encoding level (1), with a magnitude of 4, the table number is incremented again because level (1) is greater than the second threshold (which is 3). Note that the final level (-2) uses a different code from the first encoded level (also -2).

Decoding:

Code	Element	Value	Output array
0000000110	coeff_token	TotalCoeffs=5, T1s=1	Empty
1	T1 sign	-	<u>-1</u>
0001	Level	-2 decoded as -3	<u>-3</u> , -1
0010	Level	+3	<u>+3</u> , -3, -1
00010	Level	+4	<u>+4</u> , 3, -3, -1
111	Level	-2	<u>-2</u> , 4, 3, -3, -1
0011	TotalZeros	2	-2, 4, 3, -3, -1
00	run_before	2	-2, 4, 3, -3, <u>0</u> , <u>0</u> , -1

All zeros have now been decoded and so the output array is:
-2, 4, 3, -3, 0, 0, -1



Table – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC == -1
0	0	1	11	1111	0000 11	01
0	1	0001 01	0010 11	0011 11	0000 00	0001 11
1	1	01	10	1110	0000 01	1
0	2	0000 0111	0001 11	0010 11	0001 00	0001 00
1	2	0001 00	0011 1	0111 1	0001 01	0001 10
2	2	001	011	1101	0001 10	001
0	3	0000 0011 1	0000 111	0010 00	0010 00	0000 11
1	3	0000 0110	0010 10	0110 0	0010 01	0000 011
2	3	0000 101	0010 01	0111 0	0010 10	0000 010
3	3	0001 1	0101	1100	0010 11	0001 01
0	4	0000 0001 11	0000 0111	0001 111	0011 00	0000 10
1	4	0000 0011 0	0001 10	0101 0	0011 01	0000 0011
2	4	0000 0101	0001 01	0101 1	0011 10	0000 0010
3	4	0000 11	0100	1011	0011 11	0000 000
0	5	0000 0000 111	0000 0100	0001 011	0100 00	-
1	5	0000 0001 10	0000 110	0100 0	0100 01	-
2	5	0000 0010 1	0000 101	0100 1	0100 10	-
3	5	0000 100	0011 0	1010	0100 11	-
0	6	0000 0000 0111 1	0000 0011 1	0001 001	0101 00	-
1	6	0000 0000 110	0000 0110	0011 10	0101 01	-
2	6	0000 0001 01	0000 0101	0011 01	0101 10	-
3	6	0000 0100	0010 00	1001	0101 11	-



Table 9-6 – Codeword table for level_prefix

level_prefix bit string

0	1
1	01
2	001
3	0001
4	0000 1
5	0000 01
6	0000 001
7	0000 0001
8	0000 0000 1
9	0000 0000 01
10	0000 0000 001
11	0000 0000 0001
12	0000 0000 0000 1
13	0000 0000 0000 01
14	0000 0000 0000 001
15	0000 0000 0000 0001



Table – total_zeros tables for 4x4 blocks with TotalCoeff(coeff_token) 1 to 7

total_zeros	TotalCoeff(coeff_token)						
	1	2	3	4	5	6	7
0	1	111	0101	0001 1	0101	0000 01	0000 01
1	011	110	111	111	0100	0000 1	0000 1
2	010	101	110	0101	0011	111	101
3	0011	100	101	0100	111	110	100
4	0010	011	0100	110	110	101	011
5	0001 1	0101	0011	101	101	100	11
6	0001 0	0100	100	100	100	011	010
7	0000 11	0011	011	0011	011	010	0001
8	0000 10	0010	0010	011	0010	0001	001
9	0000 011	0001 1	0001 1	0010	0000 1	001	0000 00
10	0000 010	0001 0	0001 0	0001 0	0001	0000 00	
11	0000 0011	0000 11	0000 01	0000 1	0000 0		
12	0000 0010	0000 10	0000 1	0000 0			
13	0000 0001 1	0000 01	0000 00				
14	0000 0001 0	0000 00					
15	0000 0000 1						



Table – Tables for run_before

run_before	zerosLeft						
	1	2	3	4	5	6	>6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	001	101
3	-	-	00	001	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8	-	-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001



Example2

4x4 block:

0	3	-1	0
0	-1	1	0
1	0	0	0
0	0	0	0

Reordered block:

0,3,0,1,-1,-1,0,1,0...

TotalCoeffs = 5 (indexed from highest frequency [4] to lowest frequency [0])

TotalZeros = 3

T1s = 3 (in fact there are 4 trailing ones but only 3 can be encoded as a “special case”)

Example2

Encoding:

Element	Value	Code
coeff_token	TotalCoeffs=5, T1s=3	0000100
T1 sign (4)	+	0
T1 sign (3)	-	1
T1 sign (2)	-	1
Level (1)	+1 (use Level_VLC0)	1
Level (0)	+3 (use Level_VLC1)	0010
TotalZeros	3	111
run_before(4)	ZerosLeft=3; run_before=1	10
run_before(3)	ZerosLeft=2; run_before=0	1
run_before(2)	ZerosLeft=2; run_before=0	1
run_before(1)	ZerosLeft=2; run_before=1	01
run_before(0)	ZerosLeft=1; run_before=1	No code required; last coefficient.

The transmitted bitstream for this block is 000010001110010111101101 .

Example2

The output array is “built up” from the decoded values as shown below. Values in the output array at each stage are underlined.

Code	Element	Value	Output array
0000100	coeff_token	TotalCoeffs=5, T1s=3	Empty
0	T1 sign	+	<u>1</u>
1	T1 sign	-	<u>-1</u> , 1
1	T1 sign	-	<u>-1</u> , -1, 1
1	Level	+1	<u>1</u> , -1, -1, 1
0010	Level	+3	<u>3</u> , 1, -1, -1, 1
111	TotalZeros	3	3, 1, -1, -1, 1
10	run_before	1	3, 1, -1, -1, <u>0</u> , 1
1	run_before	0	3, 1, -1, -1, 0, 1
1	run_before	0	3, 1, -1, -1, 0, 1
01	run_before	1	3, <u>0</u> , 1, -1, -1, 0, 1

The decoder has inserted two zeros; however, TotalZeros is equal to 3 and s before the lowest coefficient, making the final output array:

0, 3, 0, 1, -1, -1, 0, 1

(3)、CABAC 编码

- 算术编码是把整个信源表示为实数中**0~1**之间的一个区间，其长度等于该序列的概率；
- 在区间内选择一个代表性的小数，转化为二进制作为实际的编码输出；
- 算术编码的平均编码长度为小数；

小 结

- **H.264**优点;
- **H.264**的**Profile**及应用
- **Slices**
- **H.264**技术特点（7个方面）
- 整数变换与**DCT**
- 量化算法改进
- **CAVLC**

作业6（4月14日）

- 1、说明 H.264的整数变换与DCT变换的相同点和不同点，根据压缩数据类型不同，H.264有哪几种不同的变换方式？
- 2、H.264的量化表有什么特点？具体的量化运算过程有什么特点？如何实现的？
- 3、H.264的 4×4 残差块系数矩阵为 \mathbf{X} ，求整数变换后的系数矩阵 \mathbf{Y} 。

$$\mathbf{X} = \begin{pmatrix} 2 & 1 & 0 & 1 \\ 3 & 1 & 2 & 2 \\ 1 & 2 & 4 & 1 \\ 1 & 4 & 3 & 3 \end{pmatrix}$$

作业6（4月14日）

- 4、什么是CAVLC？CAVLC编码的基本思路是什么？它利用了残差数据块的哪些特性？
- 5、试对下面整数变换后的残差块进行CAVLC编码（NC=1,写出过程及结果）。

$$\begin{pmatrix} 0 & -3 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- 6、H.264编码标准的整数变换有什么特点？