





第3章 数字视频编码原理



本章学习目标

- 了解图像和视频编码技术的发展历程，熟悉视频编码的各种方法。
- 重点掌握哈夫曼编码、算术编码、预测编码和基于**DCT**的变换编码基本原理。
- 掌握运动估计和运动补偿预测编码的基本原理。



第3章 数字视频编码原理

- 3.1 数字视频编码概述
- 3.2 熵编码
- 3.3 预测编码
- 3.4 变换编码



3.1.1 数字视频压缩的必要性和可能性

- 数据压缩的理论基础是信息论。从信息论的角度来看，压缩就是**去掉数据中的冗余**，即保留不确定的信息，去掉确定的信息（可推知的），也就是用一种更接近信息本质的描述来代替原有冗余的描述。
- 在一般的图像和视频数据中，主要存在以下几种形式的冗余。

3.1.1 数字视频压缩的必要性和可能性

- **空间冗余：**也称为空域冗余，是一种与像素间相关性直接联系的数据冗余。

例：图像中包含许多规则物体，它们的亮度、饱和度及颜色可能都一样，因此，图像在空间上具有很强的相关性。例如 Lenna 图像的脸部和肩部。



3.1.1 数字视频压缩的必要性和可能性

- **时间冗余：**也称为时域冗余，它是针对视频序列图像而言的。

视频序列每秒有25 ~ 30帧图像，相邻帧之间的时间间隔很小；同时实际生活中的运动物体具有运动一致性，使得视频序列图像之间有很强的相关性。



3.1.1 数字视频压缩的必要性和可能性

■ 统计冗余

- **信源熵**：如果将信源所有可能事件的信息量进行平均，就得到了信源熵(entropy)。熵就是平均信息量。

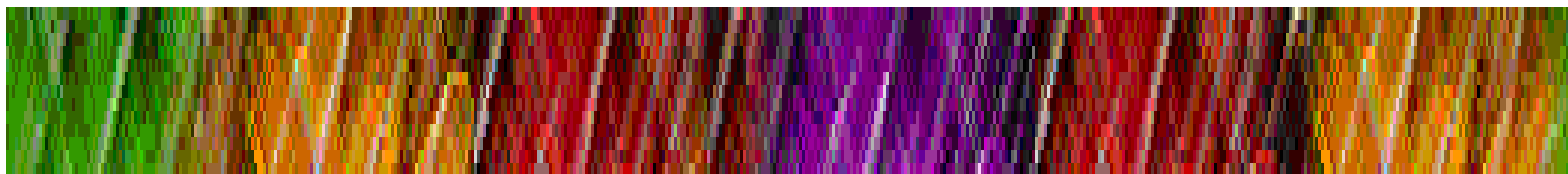
$$H(X) = E\{I(x_j)\} = \sum_{j=1}^n P(x_j) \cdot I(x_j) = -\sum_{j=1}^n P(x_j) \cdot \log_2 P(x_j)$$

- 当 x_j 等概率时， $H(X)$ 最大。
- 当 x_j 非等概率时， $H(X)$ 不是最大，就存在冗余。

采用可变长编码技术，对出现概率大的符号用短码字表示，对出现概率小的符号用长码字表示，则可去除符号冗余，从而节约码字，这就是**熵编码**的思想。

3.1.1 数字视频压缩的必要性和可能性

- **结构冗余：**在有些图像的部分区域内有着很相似的纹理结构，或是图像的各个部分之间存在着某种关系，例如自相似性等，这些都是结构冗余的表现。

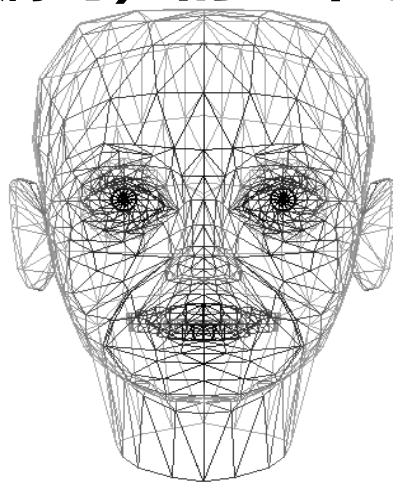


分形图像编码的基本思想就是利用了结构的自相似性。

3.1.1 数字视频压缩的必要性和可能性

- **知识冗余：**在某些特定的应用场合，编码对象中包含的信息与某些先验的基本知识有关。例如：人脸的图像有同样的结构：嘴的上方有鼻子，鼻子上方有眼睛，鼻子在中线上……

可以利用这些先验知识为编码对象建立模型。通过提取模型参数，对**参数**进行编码而不是对图像像素值直接进行编码，可以达到非常高的压缩比。这是**模型基编码**（或称**知识基编码**、**语义基编码**）的基本思想。





3.1.1 数字视频压缩的必要性和可能性

- 人眼的视觉冗余

视觉冗余度是相对于人眼的视觉特性而言的。压缩视觉冗余的核心思想是去掉那些相对人眼而言是看不到的或可有可无的图像数据。对视觉冗余的压缩通常反映在各种具体的压缩编码过程中。



3.1.2 数字视频编码技术的进展

- 1948年提出电视信号的数字化，人们开始了对图像压缩编码的研究工作。
- 1952年哈夫曼给出最优变长码的构造方法。



3.1.2 数字视频编码技术的进展

■ 预测编码

- ✧ 1952年，贝尔实验室的奥利弗等人开始研究**线性预测编码理论**
- ✧ 1958年，格雷哈姆用计算机模拟法研究图像的**DPCM方法**
- ✧ 1966年，奥尼尔通过理论分析和计算模拟比较了PCM和DPCM对电视信号进行编码传输的性能
- ✧ 20世纪70年代开始进行了**帧间预测编码**的研究
- ✧ 20世纪80年代初开始对作运动补偿预测所用的**运动估值**进行研究



3.1.2 数字视频编码技术的进展

■ 变换编码

- ❧ 首先讨论了包括K-L (Karhunen-Loeve) 变换、傅立叶变换等正交变换
- ❧ 1968年安德鲁斯等人采用二维离散傅立叶变换 (2D-DFT) 提出了变换编码
- ❧ 此后相继出现了沃尔什-哈达玛 (Walsh-Hadamard) 变换、斜 (Slant) 变换、K-L变换、离散余弦变换 (DCT) 等



3.1.2 数字视频编码技术的进展

■ 子带编码

- ❧ 1976年美国贝尔系统的克劳切等人提出了话音的子带编码。
- ❧ 1985年奥尼尔将子带编码引入到图像编码。



3.1.2 数字视频编码技术的进展

■ 算术编码

- ❧ 1960年，P.Elias提出了算术编码的概念。
- ❧ 1976年，R.Pasco和J.Rissanen分别用定长的寄存器实现了有限精度的算术编码。
- ❧ 1979年Rissanen和G.G.Langdon一起将算术编码系统化，并于1981年实现了二进制编码。
- ❧ 1987年Witten等人发表了一个实用的算术编码程序，即CACM87（后被ITU-T的H.263视频压缩标准采用）。
- ❧ 同期，IBM公司发表了著名的Q-编码器（后被JPEG建议的扩展系统和JBIG二值图像压缩标准采用）。



3.1.2 数字视频编码技术的进展

■ 基于模型编码

✎ **1983年瑞典的Forchheimer 和Fahlander提出了基于模型编码（Model-Based Coding）的思想。**



3.1.2 数字视频编码技术的进展

■ 小波变换编码

- ❧ 1986年, Meyer在理论上证明了一维小波函数的存在。
- ❧ 1987年Mallat提出了多尺度分析的思想及多分辨率分析的概念, 提出了相应的快速小波算法——Mallat算法, 并把它有效地应用于图像分解和重构。
- ❧ 1989年, 小波变换开始用于多分辨率图像描述。



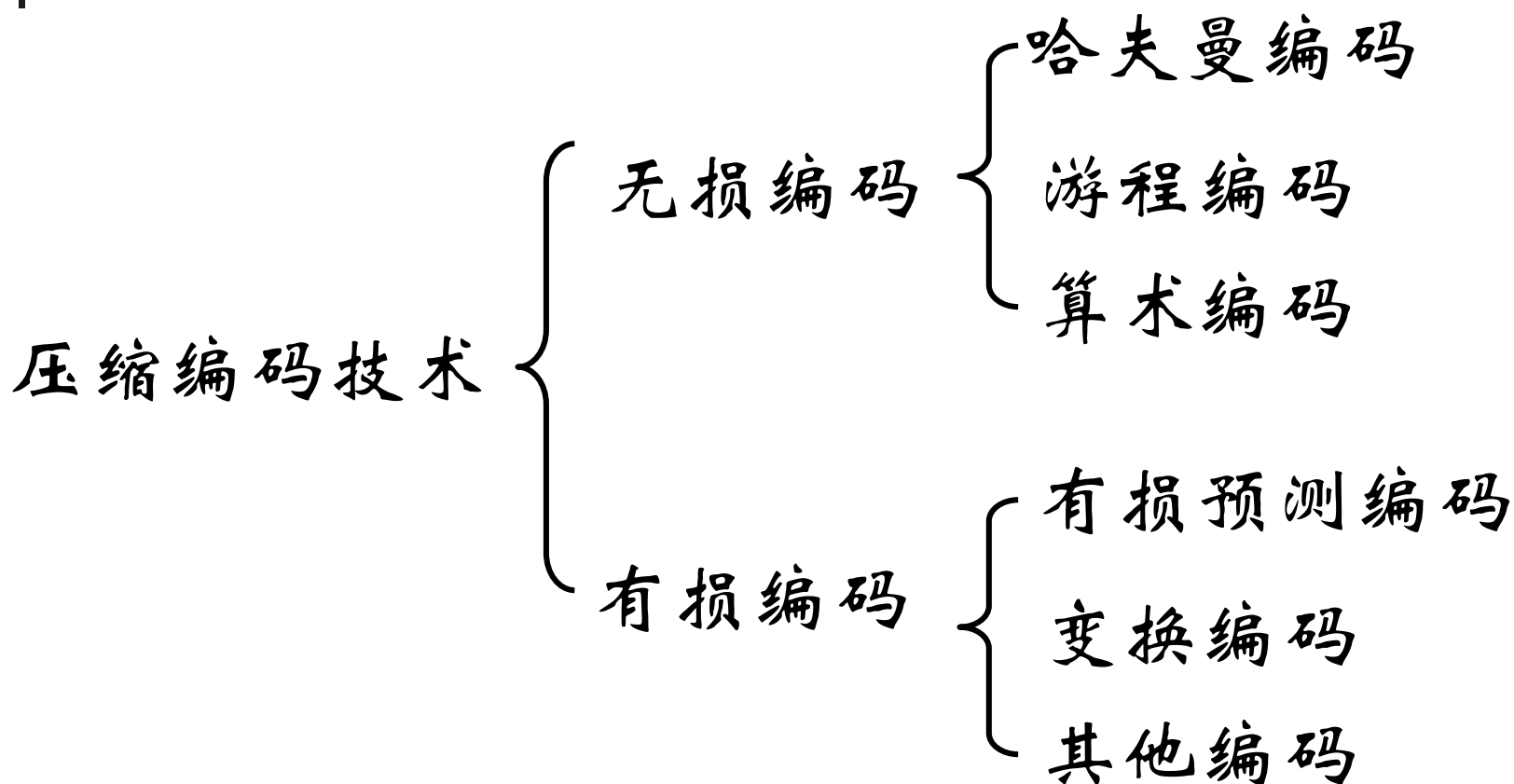
3.1.2 数字视频编码技术的进展

■ 分层可分级编码

20世纪90年代中后期，Internet迅猛发展，移动通信也迅速在全球普及，因此人们开始有了在网络上传输视频和图像的愿望。在网络上传输视频和图像等多媒体信息除了要解决误码问题之外，最大的挑战在于用户可以获得的带宽在不停地变化。为了适应网络带宽的变化，提出了分层（layered）、可分级（scalable）编码的思想。



3.1.2 数字视频编码技术的进展





3.1.2 数字视频编码技术的进展

■ 无失真编码

无失真编码又称无损编码、信息保持编码、熵编码。

熵编码是纯粹基于信号统计特性的一种编码方法，它利用信源概率分布的不均匀性，通过变长编码来减少信源数据冗余，解码后还原的数据与压缩编码前的原始数据完全相同而不引入任何失真。

无失真编码的压缩比较低，可达到的最高压缩比受到信源熵的理论限制，一般为2：1到5：1。

最常用的无失真编码方法有哈夫曼(Huffman)编码、算术编码和游程编码(Run-Length Encoding, RLE)等。



3.1.2 数字视频编码技术的进展

■ 限失真编码

限失真编码也称有损编码、非信息保持编码、熵压缩编码。

限失真编码方法利用了人类视觉的感知特性，允许压缩过程中损失一部分信息，虽然在解码时不能完全恢复原始数据，但是如果把失真控制在视觉阈值以下或控制在可容忍的限度内，则不影响人们对图像的理解，却换来了高压缩比。在限失真编码中，允许的失真愈大，则可达到的压缩比愈高。

。

常见的限失真编码方法有：预测编码、变换编码、矢量量化、基于模型的编码等。



第3章 数字视频编码原理

- 3.1 数字视频编码概述
- **3.2 熵编码**
- 3.3 预测编码
- 3.4 变换编码



3.2 熵编码

熵编码的基本原理就是去除图像信源在空间和时间上的相关性，去除图像信源像素值的概率分布不均匀性，使编码码字的平均码长接近信源的熵而不产生失真。由于这种编码完全基于图像的统计特性，因此，有时也称其为统计编码。

- **哈夫曼(Huffman)编码**
- **算术编码**
- **游程编码(Run-Length Encoding, RLE)**



3.2.1 哈夫曼编码

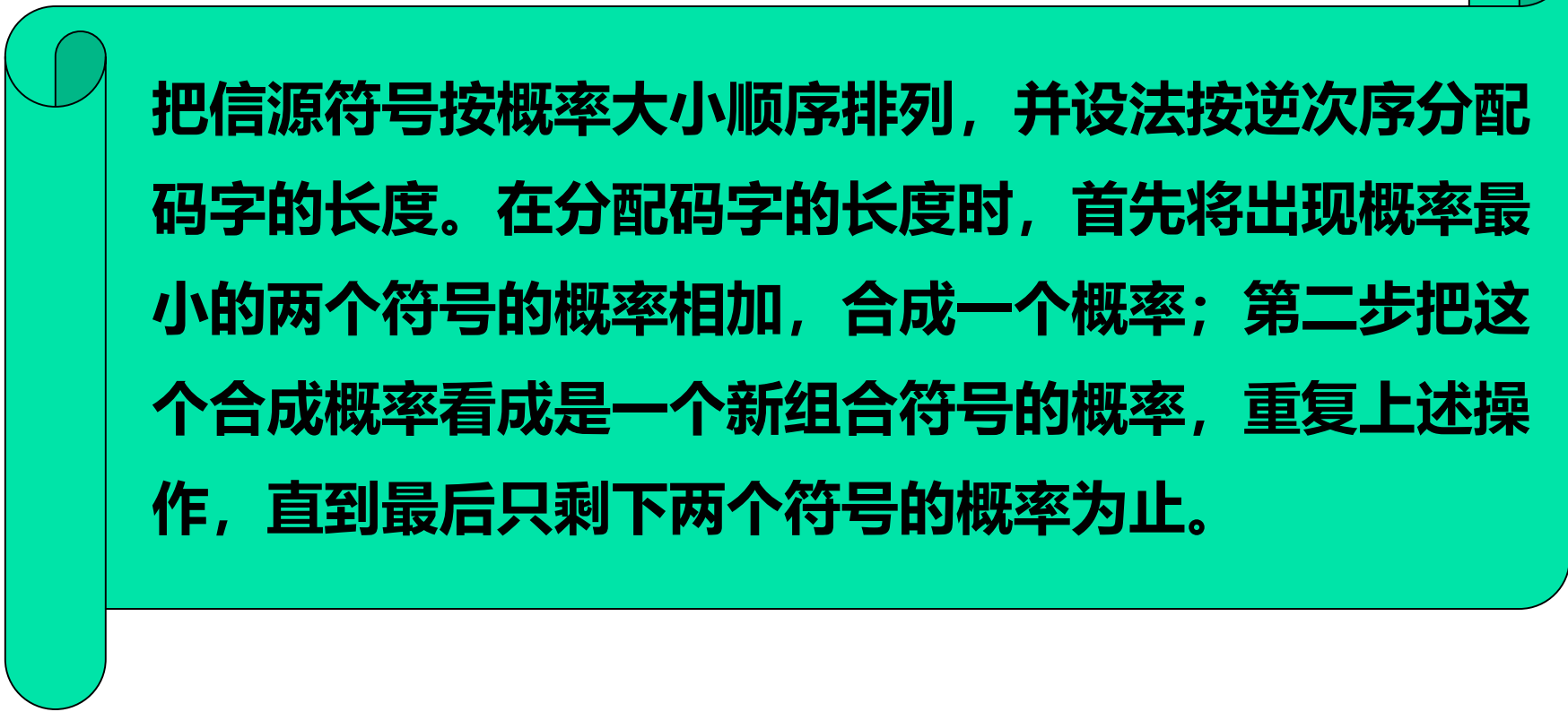
哈夫曼(Huffman)于1952年提出一种编码方法，完全依据符号出现概率来构造异字头（前缀）的平均长度最短的码字，有时称之为最佳编码。

哈夫曼编码是一种可变长度编码（Variable Length Coding, VLC），各符号与码字一一对应，是一种分组码。



3.2.1 哈夫曼编码

- Huffman编码过程 (1)



把信源符号按概率大小顺序排列，并设法按逆次序分配码字的长度。在分配码字的长度时，首先将出现概率最小的两个符号的概率相加，合成一个概率；第二步把这个合成概率看成是一个新组合符号的概率，重复上述操作，直到最后只剩下两个符号的概率为止。



3.2.1 哈夫曼编码

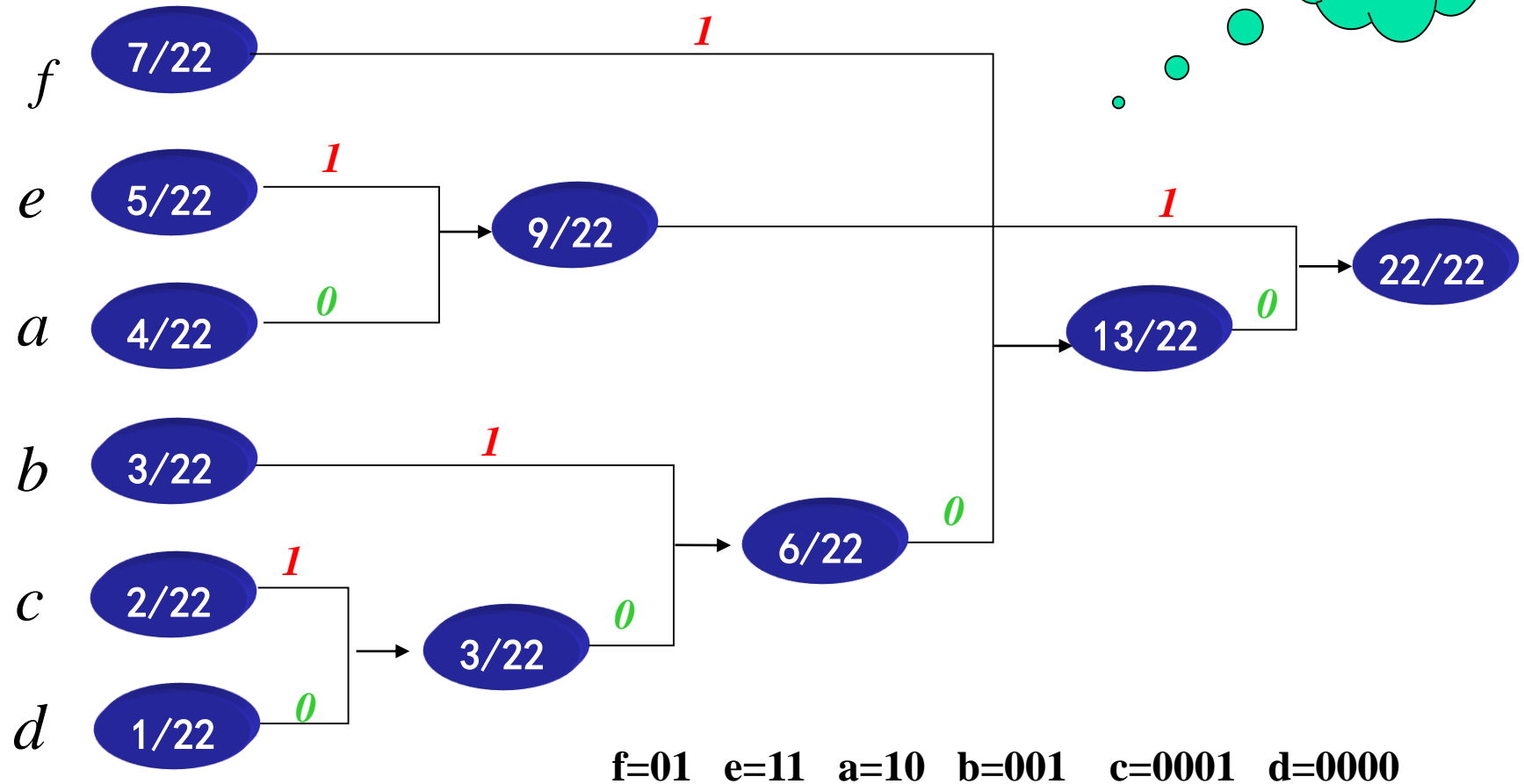
- Huffman编码过程 (2)

完成以上概率相加顺序排列后，再反过来逐步向前进行编码，每一步有两个分支，各赋予一个二进制码，可以对概率大的编码赋予0，概率小的编码赋予1。反之，也可以对概率大的编码赋予1，概率小的编码赋予0。

Huffman编码举例

aaaa bbb cc d eeeeee ffffff

编码
过程





● Huffman编码举例

输入	输入概率
----	------

S_1	0.4
-------	-----

S_2	0.3
-------	-----

S_3	0.1
-------	-----

S_4	0.1
-------	-----

S_5	0.06
-------	------

S_6	0.04
-------	------



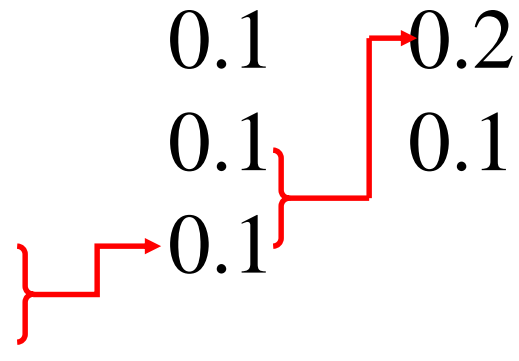
● Huffman编码举例

输入	输入概率	第一步
S_1	0.4	0.4
S_2	0.3	0.3
S_3	0.1	0.1
S_4	0.1	0.1
S_5	0.06	0.1
S_6	0.04	



● Huffman编码举例

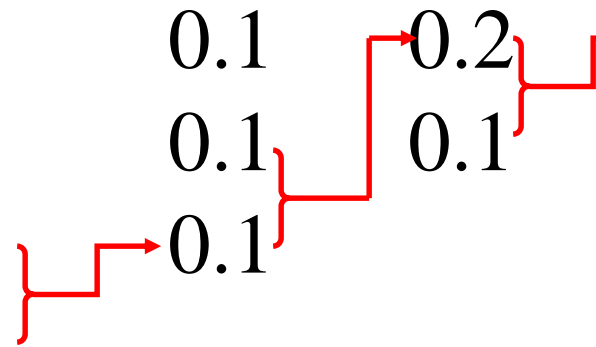
输入	输入概率	第一步	第二步
S_1	0.4	0.4	0.4
S_2	0.3	0.3	0.3
S_3	0.1	0.1	0.2
S_4	0.1	0.1	0.1
S_5	0.06	0.1	
S_6	0.04		





● Huffman编码举例

输入	输入概率	第一步	第二步	第三步
S_1	0.4	0.4	0.4	0.4
S_2	0.3	0.3	0.3	0.3
S_3	0.1	0.1	0.2	0.3
S_4	0.1	0.1	0.1	
S_5	0.06	0.1		
S_6	0.04			



● Huffman编码举例

输入	输入概率	第一步	第二步	第三步	第四步
S_1	0.4	0.4	0.4	0.4	0.6
S_2	0.3	0.3	0.3	0.3	0.4
S_3	0.1	0.1	0.2	0.3	
S_4	0.1	0.1	0.1		
S_5	0.06	0.1			
S_6	0.04				

● Huffman编码举例

输入	输入概率	第一步	第二步	第三步	第四步
S_1	0.4	0.4	0.4	0.4	0.6 0
S_2	0.3	0.3	0.3	0.3	0.4 1
S_3	0.1	0.1	0.2	0.3	
S_4	0.1	0.1	0.1		
S_5	0.06	0.1			
S_6	0.04				

● Huffman编码举例

输入	输入概率	第一步	第二步	第三步	第四步
S_1	0.4	0.4	0.4	0.4	0.6 0
S_2	0.3	0.3	0.3	0.3	0.4 1
S_3	0.1	0.1	0.2	0.3	
S_4	0.1	0.1	0.1		
S_5	0.06	0.1			
S_6	0.04				

$S_1=1$

● Huffman编码举例

输入	输入概率	第一步	第二步	第三步	第四步
S_1	0.4	0.4	0.4	0.4	0.6 0
S_2	0.3	0.3	0.3	0.3	0.4 1
S_3	0.1	0.1	0.2	0.3	
S_4	0.1	0.1	0.1		
S_5	0.06	0.1			
S_6	0.04				

$S_2=00$

● Huffman编码举例

输入	输入概率	第一步	第二步	第三步	第四步
S_1	0.4	0.4	0.4	0.4	0.6 0
S_2	0.3	0.3	0.3	0.3	0.4 1
S_3	0.1	0.1	0.2	0.3	
S_4	0.1	0.1	0.1		
S_5	0.06	0.1			
S_6	0.04				

$S_3=011$

● Huffman编码举例

输入	输入概率	第一步	第二步	第三步	第四步
S_1	0.4	0.4	0.4	0.4	0.6 0
S_2	0.3	0.3	0.3	0.3	0.4 1
S_3	0.1	0.1	0.2	0.3	
S_4	0.1	0.1	0.1		
S_5	0.06	0.1			
S_6	0.04				

$S_4=0100$

● Huffman编码举例

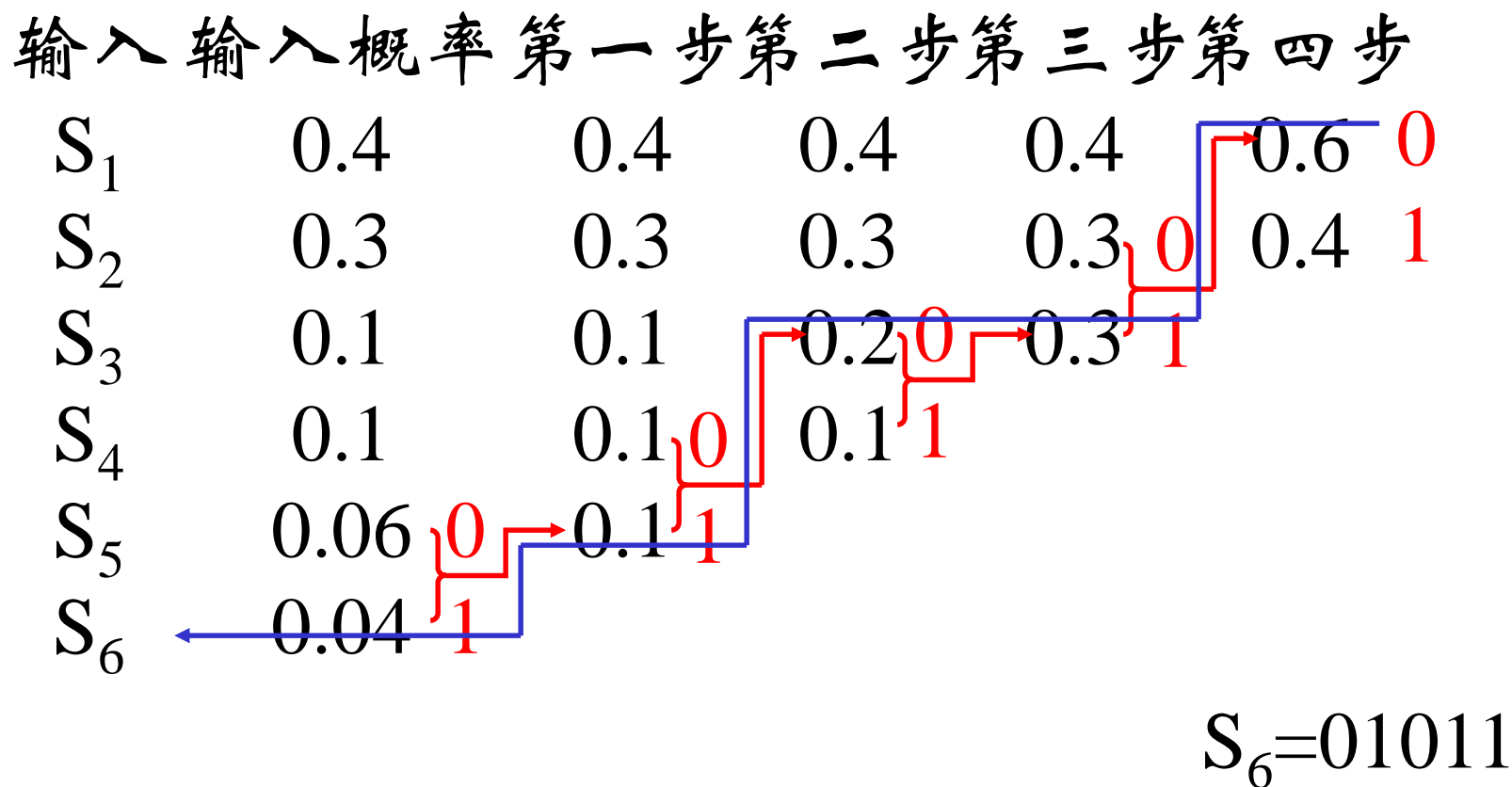
输入 输入概率 第一步 第二步 第三步 第四步

S_1	0.4	0.4	0.4	0.4	0.6	0
S_2	0.3	0.3	0.3	0.3	0.4	1
S_3	0.1	0.1	0.2	0.3		
S_4	0.1	0.1	0.1			
S_5	0.06	0.1				
S_6	0.04					

The diagram illustrates the step-by-step construction of a Huffman tree. It shows the merging of nodes S_1 through S_6 based on their probabilities. The process starts with individual nodes and their probabilities, then shows how they are combined in pairs (e.g., S_5 and S_6 merge to 0.1, then S_4 and that pair merge to 0.2, etc.) until a single root node with probability 0.6 is formed. The final column shows the resulting binary codes: 0 for S_1 and 1 for S_2 .

$S_5=01010$

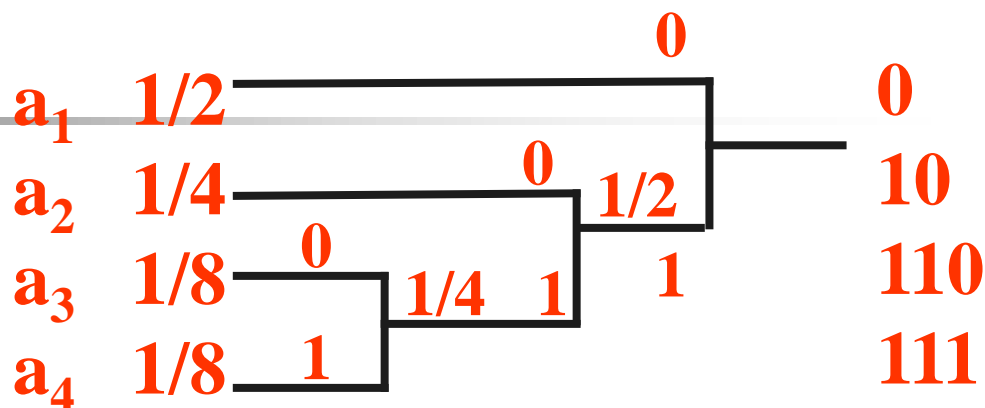
● Huffman编码举例



例：信源有四个符号：

X	a ₁	a ₂	a ₃	a ₄
	1/2	1/4	1/8	1/8

■ 信源熵：



$$H(X) = -\sum_{k=1}^K P(a_k) \cdot \log_2(P(a_k))$$

$$= -\frac{1}{2} \log_2\left(\frac{1}{2}\right) - \frac{1}{4} \log_2\left(\frac{1}{4}\right) - 2 \times \frac{1}{8} \log_2\left(\frac{1}{8}\right) = 1.75 \text{ bit / Symbol}$$

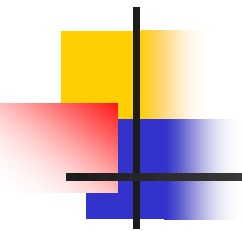
■ Huffman（二进制编码）

a ₁	a ₂	a ₃	a ₄
0	10	110	111

平均码长： $I_{av} = (1/2) \times 1 + (1/4) \times 2 + (1/8) \times 6 = 1.75 \text{ bit/字符}$

编码效率： $\eta = 1.75/1.75 = 100\%$

编码冗余： $R = 0$

- 
- 4 个符号的等长码:

$$B = \log_2 4 = 2\text{bit}$$

a_1	a_2	a_3	a_4
00	01	10	11

$$L = 2$$

编码效率: $\eta = H(X)/L = 1.75/2 = 87.5\%$

a_i	$P(a_i)$	等长码	Huffman 码
a_1	1/2	00	0
a_2	1/4	01	10
a_3	1/8	10	110
a_4	1/8	11	111

- **Huffman** 编码举例:

原信源输出序列: $a_1 a_2 a_1 a_3 a_2 a_1 a_1 a_4 \dots$

编码后的序列: 0 10 0 110 10 0 0 111 ...

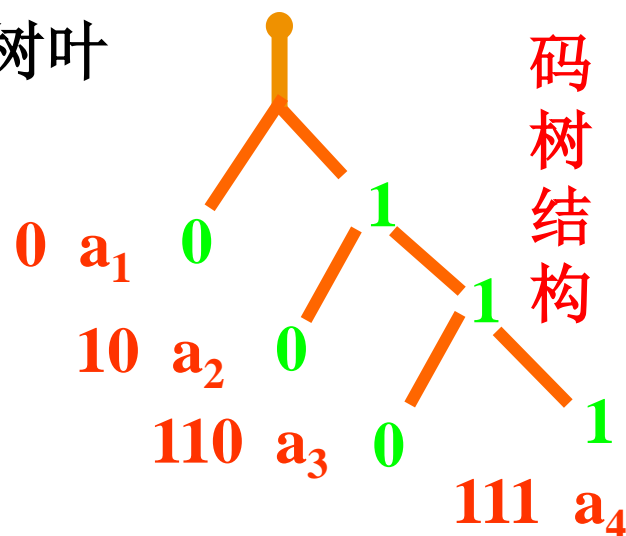
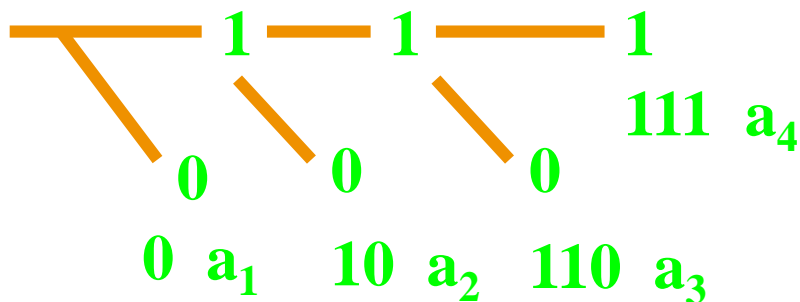
Huffman 码解码

Huffman 码是非歧义的，在解码过程中，对某个码字的解释是唯一的。

- 原信源输出序列： $a_1 a_2 a_1 a_3 a_2 a_1 a_1 a_4 \dots$ ，编码后的序列： $0\ 10\ 0\ 110\ 10\ 0\ 0\ 111 \dots$

- 当接收端收到码流 $01001101000111\dots$ 时，按照码表，以 0 开头的码字只有 0，因此，解出第 1 个符号为 a_1 ；
- 去掉已解码的 0 后，码流剩下 1001101000111 ，以 1 开头，第 2 比特是 0，因此，码字是 10，因此，解出第 2 个符号为 a_2 ；
-

- 顺着码树解码，树根→树干→树枝→树叶





3.2.1 哈夫曼编码

■ 哈夫曼编码的特点

- ③ 哈夫曼编码的算法是确定的，但编出的码并非是唯一。
- ③ 由于哈夫曼编码的依据是信源符号的概率分布，故其编码效率取决于信源的统计特性。
- ③ 哈夫曼码没有错误保护功能。
- ③ 哈夫曼码是可变长度码，码字字长参差不齐，输出码率是变化的。因此，对于恒定码率信道，需要增加输出缓存器来平滑。
- ③ 对信源进行哈夫曼编码后，形成了一个哈夫曼编码表，解码时，必须参照这一哈夫曼编码表才能正确解码。

3.2.1 哈夫曼编码

概率分布为 2 的负幂次方				概率分布为均匀分布		
信源符号	出现概率	哈夫曼码字	码字长度	出现概率	哈夫曼码字	码字长度
S_0	2^{-1}	1	1	0.125	111	3
S_1	2^{-2}	01	2	0.125	110	3
S_2	2^{-3}	001	3	0.125	101	3
S_3	2^{-4}	0001	4	0.125	100	3
S_4	2^{-5}	00001	5	0.125	011	3
S_5	2^{-6}	000001	6	0.125	010	3
S_6	2^{-7}	0000001	7	0.125	001	3
S_7	2^{-7}	0000000	7	0.125	000	3
编码效率	$H=1.984\ 375$	$R=1.984\ 375$	$\eta=100\%$	$H=3$	$R=3$	$\eta=100\%$



3.2.2 算术编码

- 从理论上分析，采用哈夫曼编码可以获得最佳信源字符编码效果；
- 实际应用中，由于信源字符出现的概率并非满足2的负幂次方，因此往往无法达到理论上的编码效率和压缩比。



3.2.2 算术编码

- 设字符序列 $\{x, y\}$ 对应的概率为 $\{1/3, 2/3\}$, N_x 和 N_y 分别表示字符 x 和 y 的最佳码长, 则根据信息论有:

$$N_x = -\log_2\left(\frac{1}{3}\right) = 1.58$$

$$N_y = -\log_2\left(\frac{2}{3}\right) = 0.588$$



3.2.2 算术编码

- 字符 x 、 y 的最佳码长分别为1.58bit和0.588bit;
- 这表明, 要获得最佳编码效果, 需要采用小数码字长度,这是不可能实现的;
- 即采用哈夫曼方法对 $\{x, y\}$ 的码字分别为0和1, 也就是两个符号信息的编码长度都为1。对于出现概率大的字符 y 并未能赋予较短的码字;
- 实际编码效果往往不能达到理论效率;
- 为提高编码效率, Elias等人提出了算术编码算法。

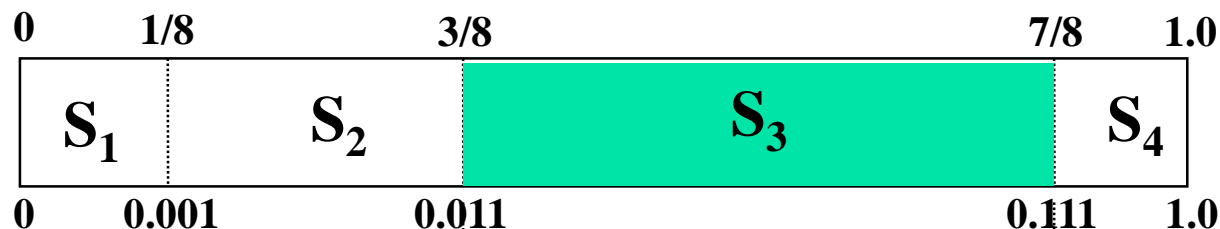


3.2.2 算术编码

算术编码是一种非分组编码，它用一个浮点数值表示**整个信源符号序列**。算术编码将被编码的信源符号序列表示成实数半开区间 $[0, 1)$ 中的一个数值间隔。这个间隔随着信源符号序列中每一个信源符号的加入逐步减小，每次减小的程度取决于当前加入的信源符号的先验概率。

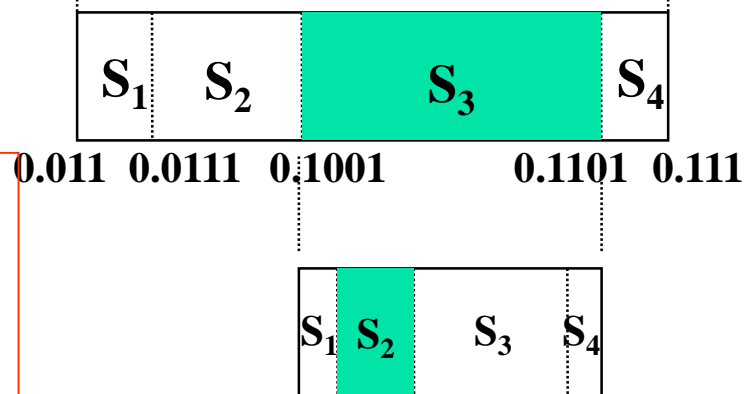
3.2.2 算术编码

符号	S_1	S_2	S_3	S_4
概率(十进制)	1/8	1/4	1/2	1/8
概率(二进制)	0.001	0.01	0.1	0.001
累积概率	0	0.001	0.011	0.111



■ 符号序列 $S_3S_3S_2S_4 \dots$ 为例

在算术编码中通常采用二进制分数表示概率，**每个符号所对应的概率区间都是半开区间**，即该区间包括左端点，而不包括右端点，如 S_1 对应 $[0, 0.001)$ ， S_2 对应 $[0.001, 0.01)$ 等。





3.2.3 游程编码

游程编码，也称行程编码或游程（行程）长度编码
(Run Length Encoding, RLE)

- **游程：具有相同灰度值的像素序列。**
- **游程长度：灰度值相同的相邻像素的数目。**
- **游程编码思想：去除像素冗余。**
 - **用游程的灰度和游程的长度代替游程本身。**

例：设重复次数为 iC , 重复像素值为 iP

编码为： $iPiC\ iPiC\ iPiC$

编码前： aaaaaaa bbbbbb cccccccc

编码后： a7b6c8

3.2.3 游程编码

- 由于一幅图像中有许多颜色相同的图块，用一整数对存储一个像素的颜色值及相同颜色像素的数目（长度）。例如：

(G , L)
颜色 长度
值

编码时采用从左到右，从上到下的排列，
每当遇到一串相同数据时就用该数据及
重复次数代替原来的数据串。

```
000000003333333333
2222222222226666666
111111111111111111
111115555555555555
888888888888888888
5555555555555553333
222222222222222222
```

(0,8) (3,10) (2,11) (6,7)
(1,18) (1,6) (5,12) (8,18)
(5,14) (3,4) (2,18)

18*7的像素颜色仅用11对数据



第3章 数字视频编码原理

- 3.1 数字视频编码概述
- 3.2 熵编码
- **3.3 预测编码**
- 3.4 变换编码



3.3 预测编码

预测编码的基本原理就是利用图像数据的相关性，利用已传输的像素值对当前需要传输的像素值进行预测，然后对当前像素的实际值与预测值的差值（即预测误差）进行编码传输，而不是对当前像素值本身进行编码传输，以去除图像数据中的空间相关冗余或时间相关冗余。



3.3.1 帧内预测编码

- **预测编码：根据某一模型，利用信号以往的样本值对新样本值进行预测，对预测误差进行编码。**
- **对于相关性较强的信号，如果建立合适的模型，预测误差的幅值将远远小于原始信号，从而可以用较少的量化级对其误差信号进行量化，得到较大的数据压缩效果。**



3.3.1 帧内预测编码

➤ 问题：能否**精确地预测**数据源输出？

答案：否

数据源是不确定的

- 几乎没有有一个实际的系统能找到可以精确预测输出的**模型**
- 能找到的**最优预测模型**是以某种**最小误差**意义下的预测模型。



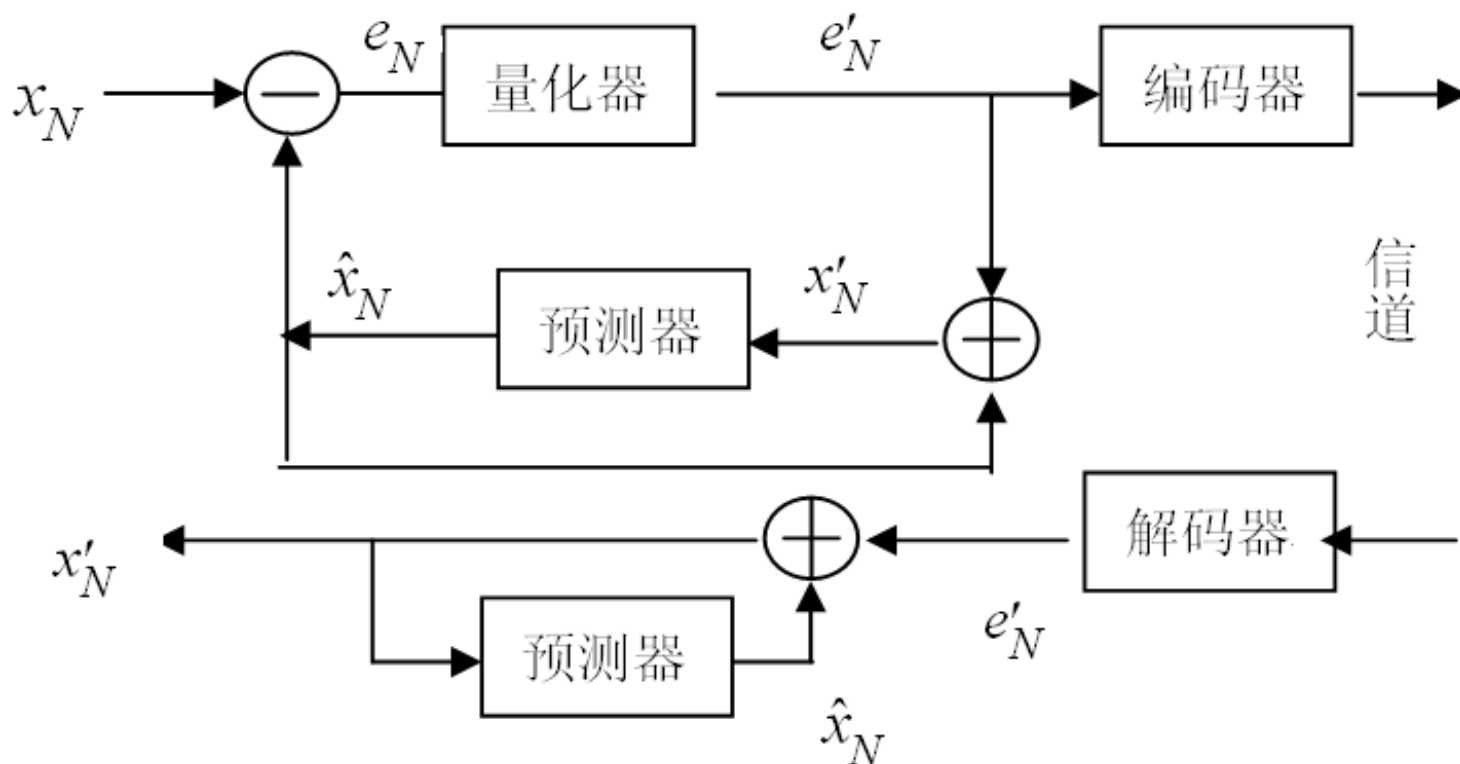
3.3.1 帧内预测编码

- **对于静止图像，由于相邻像素具有很强的相关性，这样当前像素的灰度（颜色）值可用前面已经出现的像素值进行预测，得到一个预测值，对实际值与预测值的差值进行编码，**

3.3.1 帧内预测编码

■ 1. DPCM系统的基本原理

DPCM(Differential Pulse Code Modulation, 差分脉冲编码调制)



3.3.1 帧内预测编码

■ 2. 预测模型

设 t_N 时刻之前的样本 x_1, x_2, \dots, x_{N-1} 与预测值之间的关系呈现某种函数形式

⌘ 线性预测编码器

$$\hat{x}_N = \sum_{i=1}^{N-1} a_i x_i$$

⌘ 非线性预测编码器



3.3.1 帧内预测编码

在图像数据压缩中，常用如下几种**线性预测方案**：

- * **前值预测**，即 $\hat{x}_N = x_{N-1}$
- * **一维预测**，即采用同一扫描行中前面已知的若干个样值来预测。
- * **二维预测**，即不但用同一扫描行中的前面几个样值，而且还要用以前几行扫描行中样值来预测。



3.3.2 帧间预测编码

序列图像在时间上的冗余情况可分为如下几种：

- ✧ **对于静止不动的场景**，当前帧和前一帧的图像内容是完全相同的。
- ✧ **对于运动的物体**，只要知道其运动规律，就可以从前一帧图像推算出它在当前帧中的位置。
- ✧ **摄像机**对着场景的横向移动、焦距变化等操作会引起整个图像的平移、放大或缩小。对于这种情况，只要**摄像机的运动**规律和镜头改变的参数已知，图像随时间所产生的变化也是可以推算出来的。



3.3.2 帧间预测编码

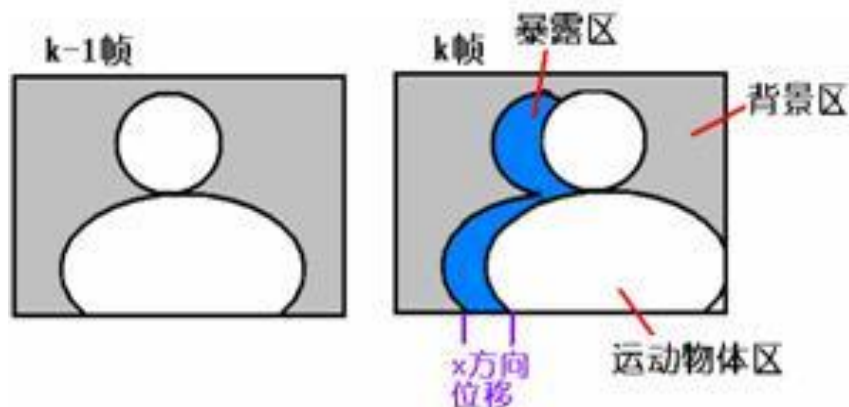
帧间预测的依据:

- ◆ 图像序列在时间轴方向的相关性;
- ◆ 物体的背景或物体的一部分相对不变或变化缓慢;
- ◆ 人类的视觉特性:
 - ◆ 人类的视觉对静止图像有较高的空间分辨率, 但是可以减少传输帧数来降低时间轴分辨率, 未传输的帧可以通过计算补出来;
 - ◆ 对运动图像分辨率低, 可以对这一部分图像降低清晰度。

3.3.2 帧间预测编码

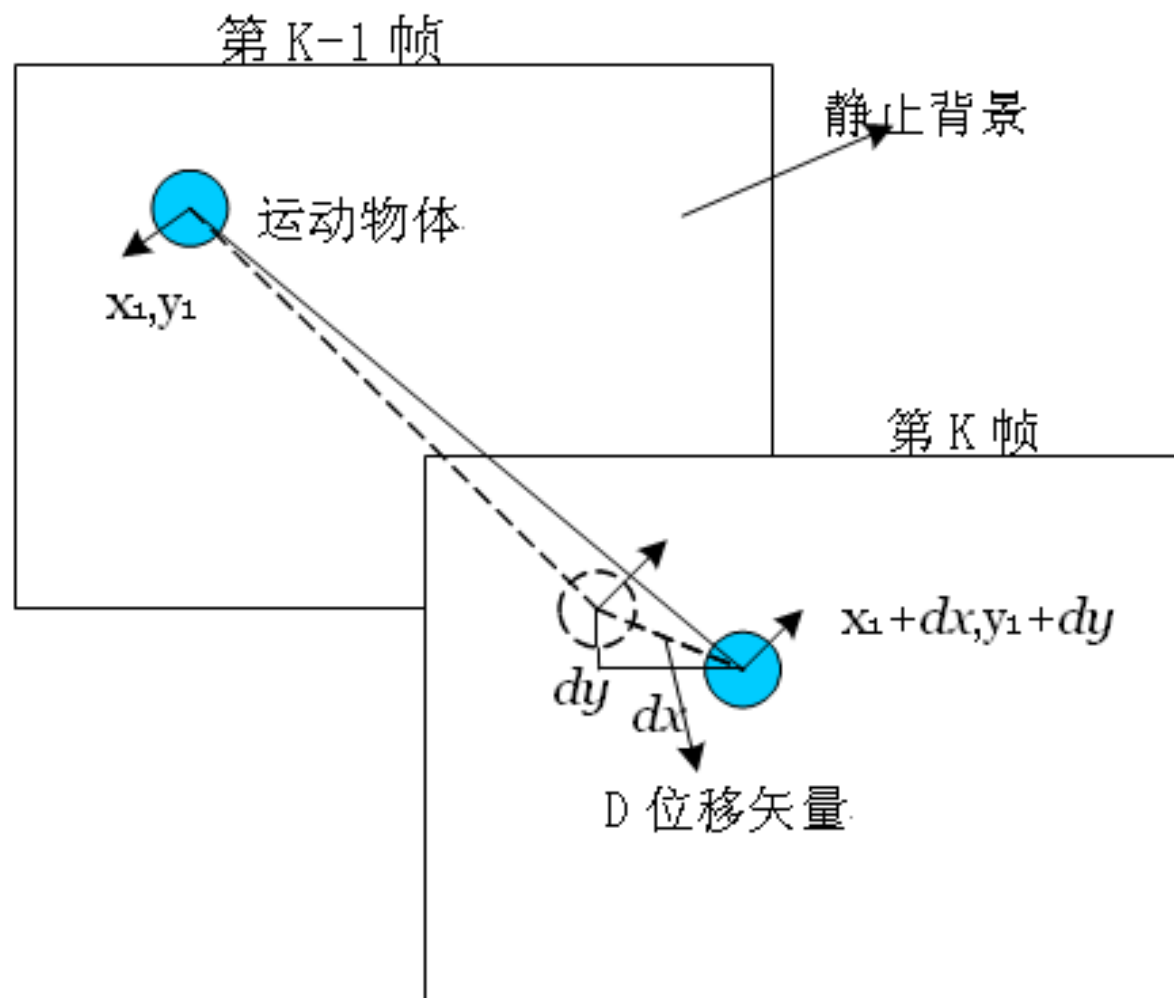
为什么进行运动补偿预测？

- ◆ 对于活动图像编码，帧间预测是主要的手段；
- ◆ 基本帧间预测方法对于存在大量静止区域或缓变区域的图像，预测效果不错；
- ◆ 对于活动的物体，预测效果不理想；
- ◆ 对于一些发生运动的图像进行预测编码，采用运动补偿预测的方法。



3.3.2 帧间预测编码

■ 运动补偿预测





3.3.2 帧间预测编码

运动补偿预测的基本原理：

- ◆ 自然场景的视频图像只有其中的部分区域在运动，同一场景相邻的两帧图像之间差异也不会太大，编码器无需将视频序列中每帧图像的所有信息都进行编码后传输给解码器端，只要将当前帧中目标的运动信息告知解码器端，解码器可根据运动信息和前一帧图像内容来更新当前帧图像，获得当前帧的真实数据。（可有效降低编码所需数据量）
 - ◆ 从序列图像中提取有关物体运动的信息的过程——**运动估计**（如何快速、有效的获得足够精度的运动矢量）；
 - ◆ 把前一帧相应的运动部分信息根据运动矢量补偿过来的过程——**运动补偿**（Motion Compensation, MC）。



3.3.2 帧间预测编码

- ◆ **运动估计**——将当前帧活动图像分为若干局部结构（像素块），检测出每个局部结构在前一帧图像中的位置，从而可以估计出这个结构的位移。即对运动物体从前一帧到当前帧位移的方向和像素数作出估计，也就是求出**运动矢量**。
- ◆ **运动补偿**——根据求出的运动矢量，找到当前帧的像素（或像素块）是从前一帧的哪个位置移动过来的，从而得到当前帧像素（或像素块）的预测值。



3.3.2 帧间预测编码

运动估计与运动补偿预测编码步骤：

- ◆ 分割图像为若干局部结构——划分静止和运动区域；
 - ◆ 最简单方法：分块
- ◆ 运动估计——对每一个运动物体进行位移估计；
- ◆ 运动补偿——由位移估计建立同一运动物体在不同帧空间位置对应关系，建立预测关系；
- ◆ 对于运动补偿后的位移帧差信号、运动矢量进行编码传输。



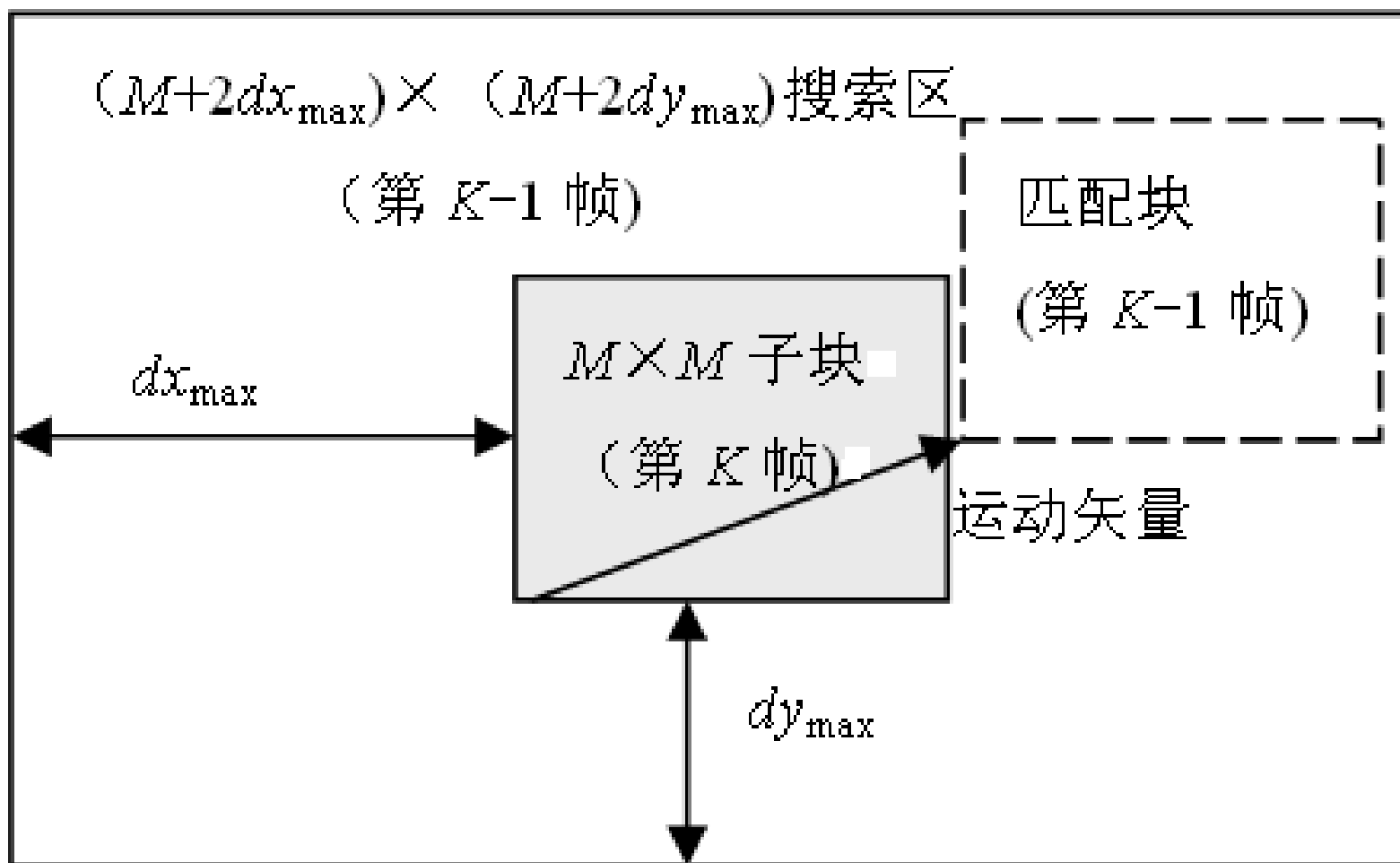
3.3.2 帧间预测编码

■ 运动估计方法：

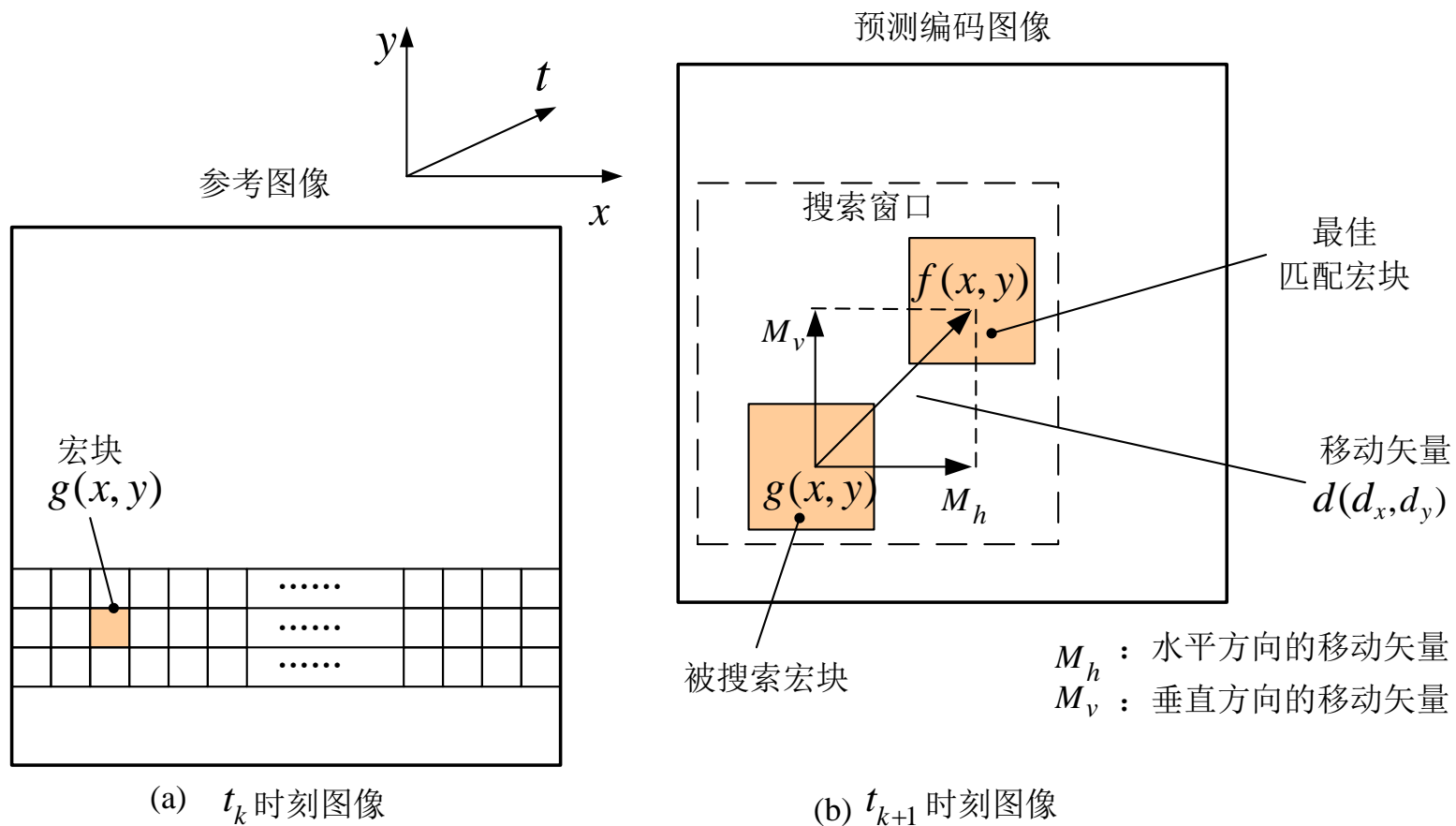
- **像素递归法**：根据像素间亮度的变化和梯度，通过递归修正的方法来估计每个像素的运动矢量。接收端在与发送端同样的条件下，用与发送端相同的方法进行运动估值。像素递归法估计精度高，可以满足运动补偿帧内插的要求。但接收端较复杂，不利于一发多收（如数字电视广播等）的应用。
- **块匹配算法**：块匹配算法对当前帧图像的每一子块，在前一帧（第 $K-1$ 帧）的一定范围内搜索最优匹配，并认为本图像子块就是从前一帧最优匹配块位置处平移过来的。块匹配算法虽然作了一定的假设（假设位于同一图像子块内的所有像素都作相同的运动，且只作平移运动），但满足了计算复杂度和实时实现的要求。

3.3.2 帧间预测编码

➤ 块匹配算法



3.3.2 帧间预测编码



运动矢量的算法框图



块匹配算法(BMA)

➤ 方块大小的选取

块大时，一个方块可能包含多个作不同运动的物体，块内各像素作相同平移运动的假设难以成立，影响估计精度。

若块太小，则估计精度容易受噪声干扰的影响，不够可靠，而且传送运动矢量所需的附加比特数过多，不利于数据压缩。

一般都用 **16×16** 像素的块作为匹配单元。



块匹配算法(BMA)

➤ 最优匹配准则

- **绝对差均值 (MAD, Mean Absolute Difference) 最小准则**

$$MAD(i, j) = \frac{1}{MM} \sum_{m=1}^M \sum_{n=1}^M |S_K(m, n) - S_{K-1}(m+i, n+j)|$$

- **均方误差 (MSE, Mean Squared Error) 最小准则**

$$MSE(i, j) = \frac{1}{MM} \sum_{m=1}^M \sum_{n=1}^M [S_K(m, n) - S_{K-1}(m+i, n+j)]^2$$

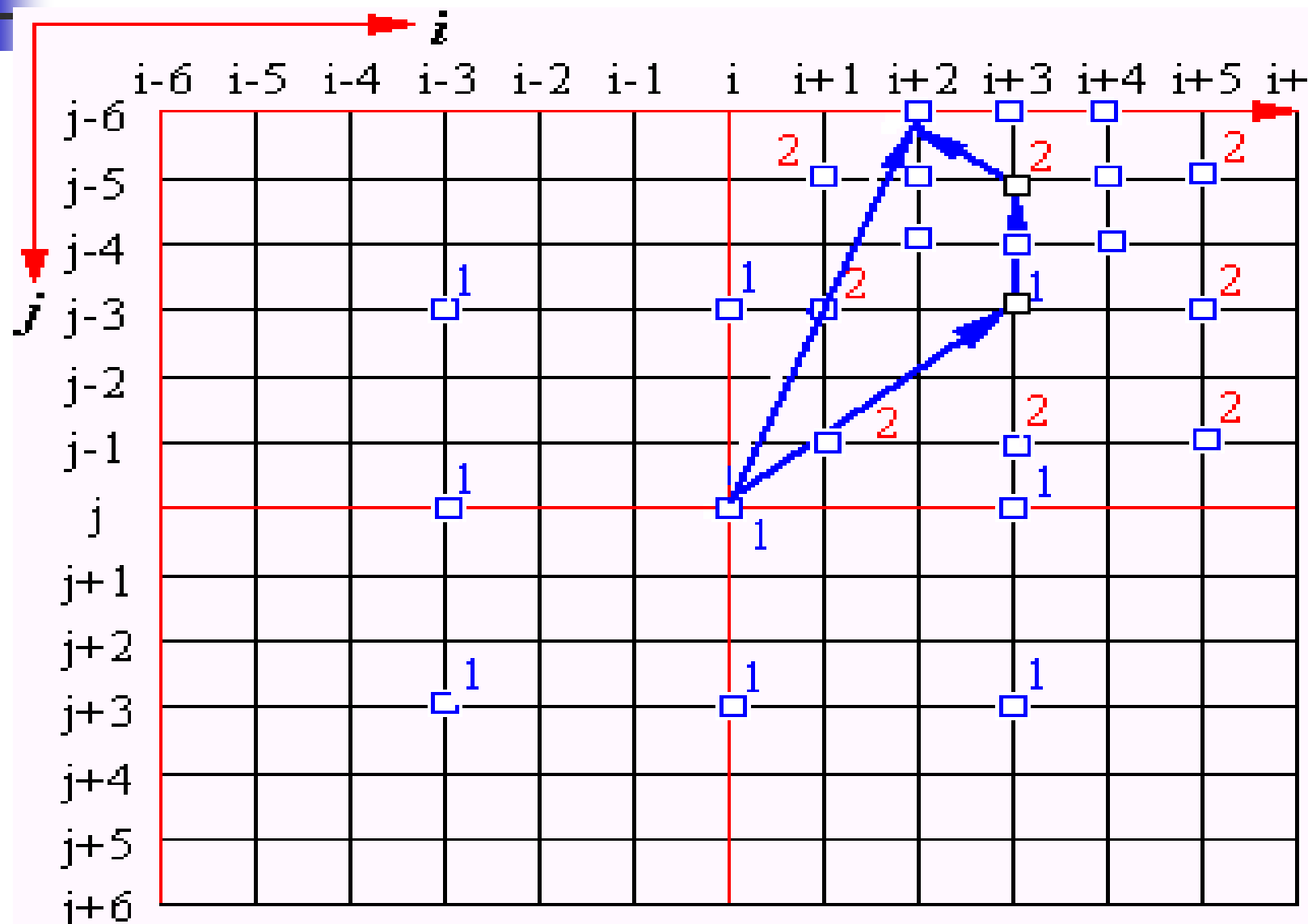
- **归一化互相关函数最大准则**

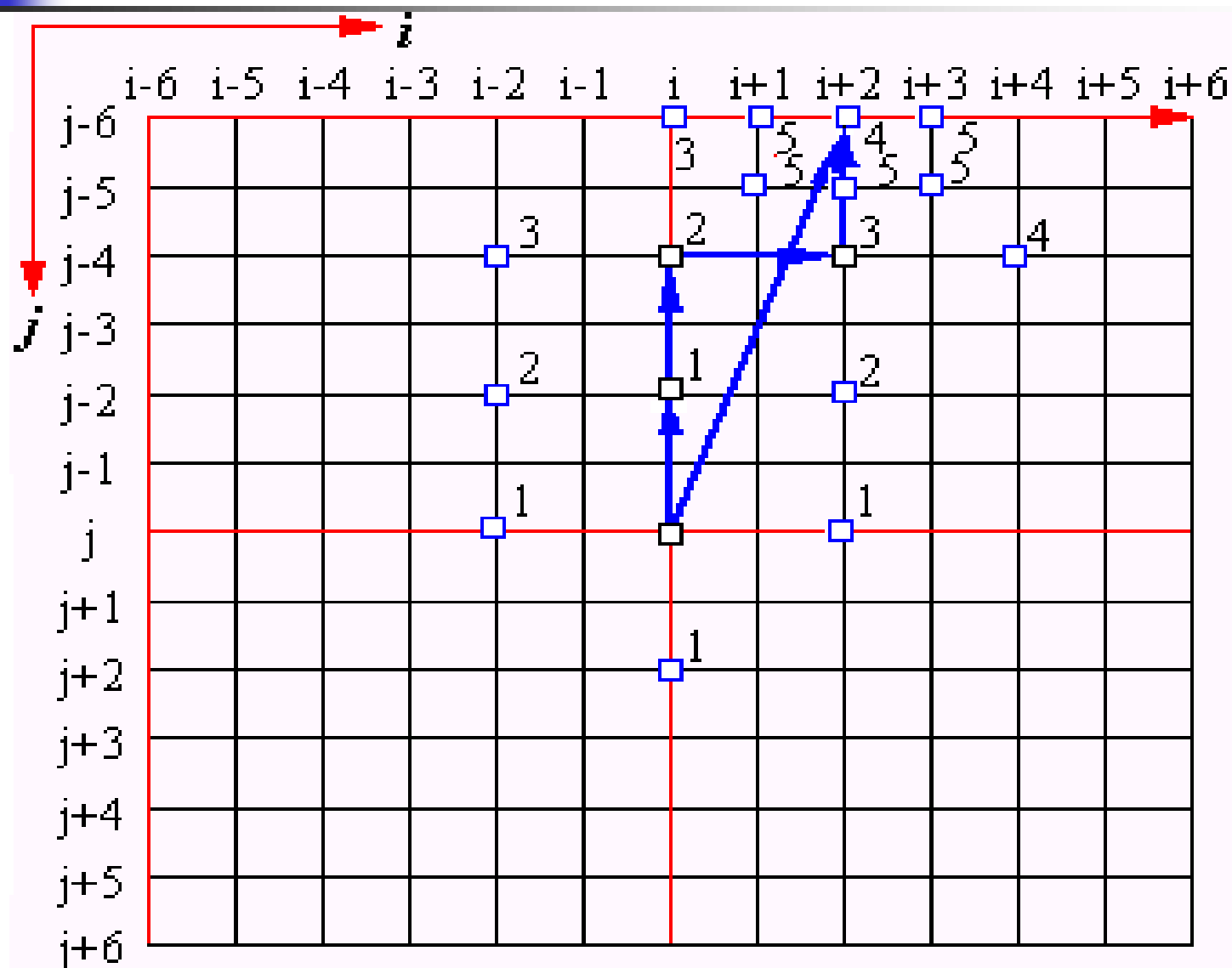


块匹配算法(BMA)

➤ 最优匹配点的搜索方法

- **穷尽搜索** (full search, 也称**全搜索**)
- **快速搜索**: 其算法共同之处在于它们把使准则函数 (例如, MAD) 趋于极小的方向视同为最小失真方向, 并假定准则函数在偏离最小失真方向时是单调递增的, 即认为它在整个搜索区内是 (i,j) 的单极点函数, 有唯一的极小值, 而快速搜索是从任一猜测点开始沿最小失真方向进行的。
- **分级搜索**: 先通过对原始图像滤波和亚采样得到一个图像序列的低分辨率表示, 再对所得低分辨率图像进行全搜索。由于分辨率降低, 使得搜索次数成倍减少, 这一步可以称为粗搜索。然后, 再以低分辨率图像搜索的结果作为下一步细搜索的起始点。经过粗、细两级搜索, 便得到了最终的运动矢量估值。



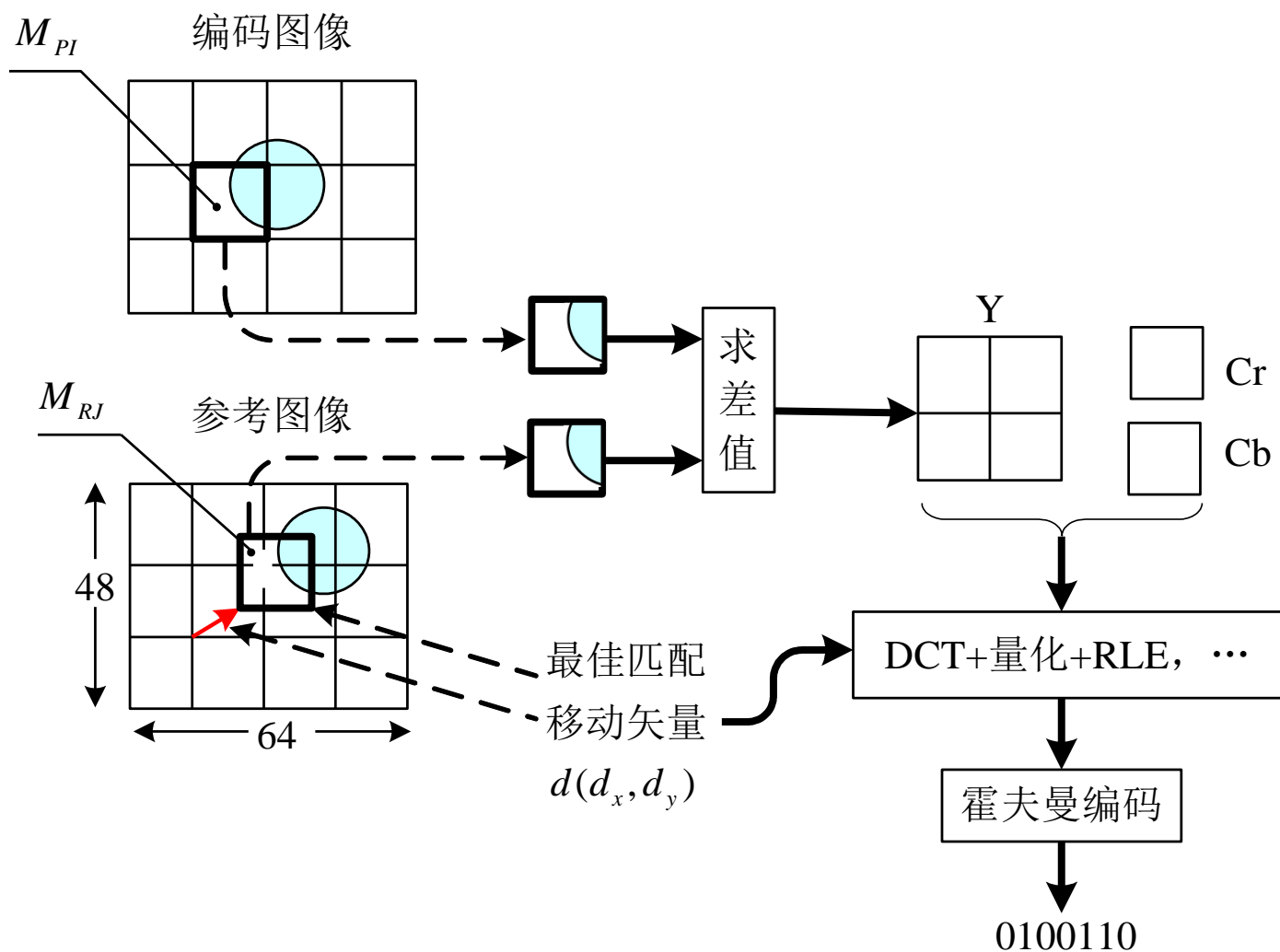




运动补偿帧间预测编码过程

- ◆ 在视频帧序列中设置参照帧，且第1帧总是参照帧。
- ◆ 对于当前的编码帧，首先在该帧的前一帧和/或后一帧（参照帧）中寻找与该帧的一个图像方块最优匹配的图像方块。
- ◆ 如果找到这样的最优匹配块，则进行下列计算：
 - 计算当前块的像素值与参照帧中最优匹配块（称参照块）的像素值之间的差值，即预测误差；
 - 计算当前块相对于参照块在水平（ x ）和垂直（ y ）两个方向上的位移，即运动矢量。
- ◆ 如果找不到最优匹配块，则必须进行帧内编码，即对当前块的像素样本值进行编码传输。

帧间预测编码原理图



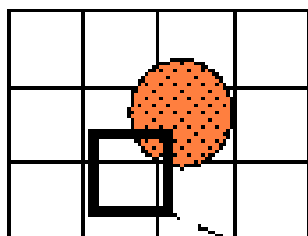


运动补偿帧间预测类型

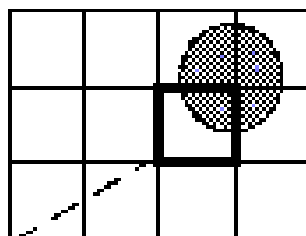
- **单向运动补偿预测：**只使用前参照帧或后参照帧中的一个来进行预测。
- **双向运动补偿预测：**使用前、后两个帧作为参照帧来计算各块的运动矢量，最后只选用与具有最小匹配误差的参照帧相关的运动矢量值。
- **插值运动补偿预测：**取前参照帧预测值与后参照帧预测值的平均值。这时，需要对两个运动矢量分别进行编码传输。

双向预测B帧的压缩编码原理

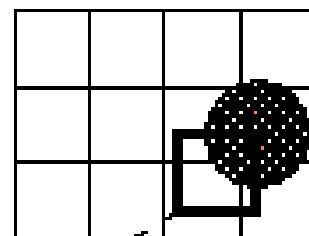
过去的参考图象



编码图象



将来的参考图象



$$\left[\text{Past Ref} - 0.5 \times \text{Current} \right] + \left[\text{Current} - 0.5 \times \text{Future Ref} \right] = \text{Residual}$$

DCT + 量化 + RLE

移动矢量

霍夫曼编码

011010...



第3章 数字视频编码原理

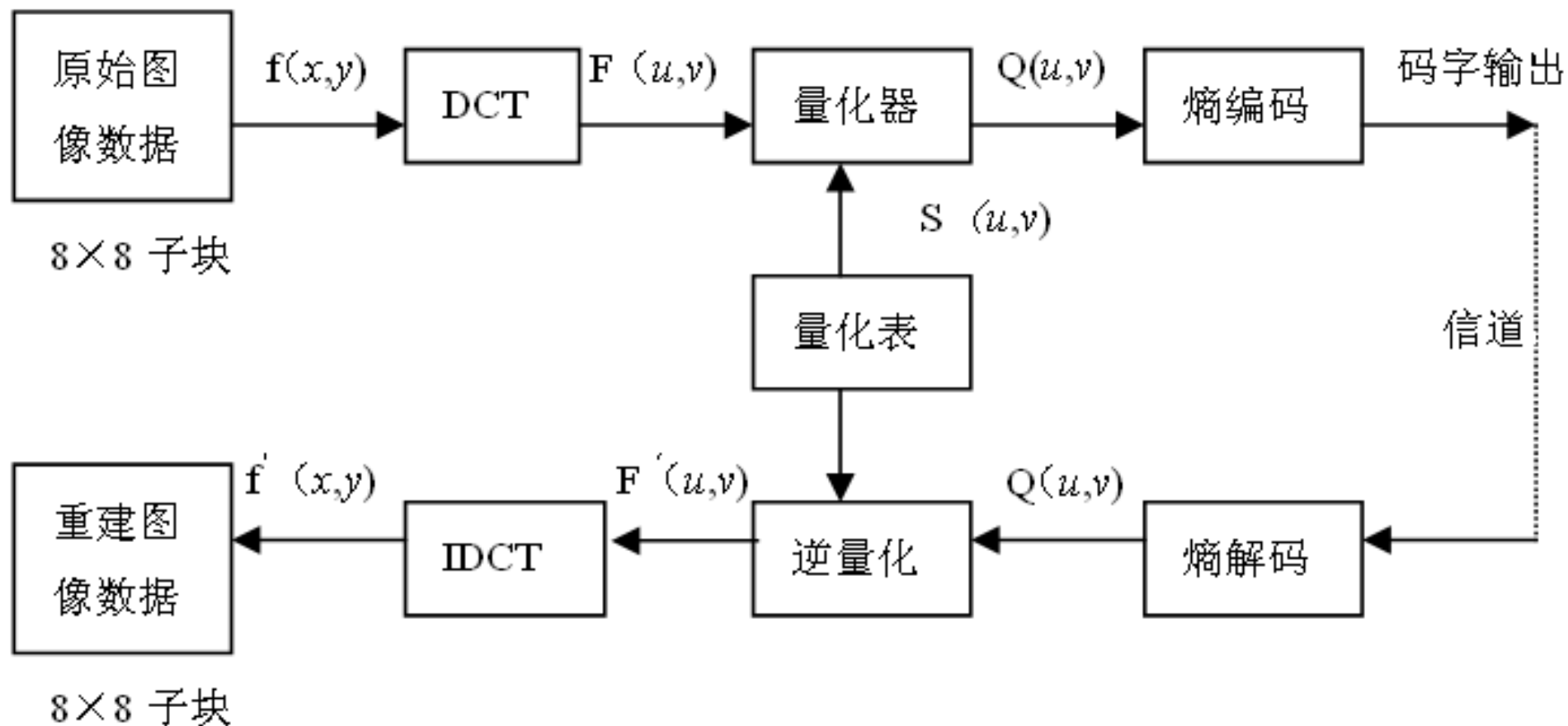
- 3.1 数字视频编码概述
- 3.2 熵编码
- 3.3 预测编码
- **3.4 变换编码**



3.4.1 变换编码的基本原理

- 预测编码希望通过对信源建模尽可能精确地预测数据，然后对预测误差进行编码。
- 变换编码的思路：将原始数据从时间域或者空间域“变换”到另一个更为紧凑表示、适合于压缩的变换域（通常为频域），从而得到比预测编码更高效的数据表示（压缩）。
- 预测编码消除相关性的能力有限，变换编码是一种更高效的压缩编码。

3.4.2 基于DCT的图像编码





3.4.2 基于DCT的图像编码

8 × 8 二维DCT变换

$$F(u, v) = \frac{1}{4} C(u) C(v) \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

8 × 8 二维DCT反变换

$$f(x, y) = \frac{1}{4} \sum_{u=0}^7 \sum_{v=0}^7 C(u) C(v) F(u, v) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

当 $u = v = 0$ 时, $C(u) = C(v) = \frac{1}{\sqrt{2}}$

当 u, v 为其他值时 $C(u) = C(v) = 1$



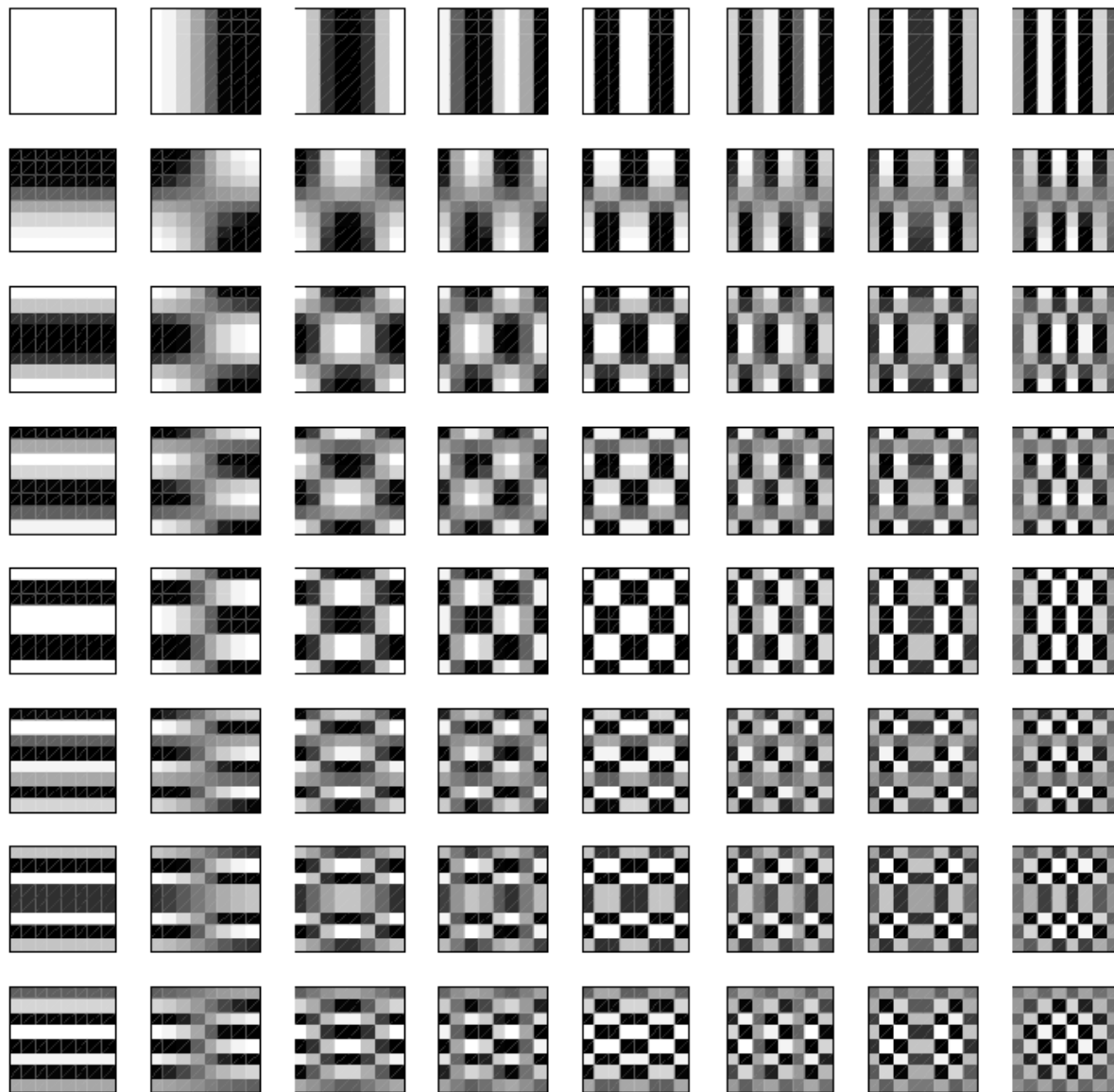
3.4.2 基于DCT的图像编码

8×8 二维DCT反变换的变换核函数为

$$C(u)C(v)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}$$

按 u ， v 分别展开后得到64个 8×8 像素的图像块组，称为基图像。

8×8二维DCT变换基图像





3.4.2 基于DCT的图像编码

量化

量化处理是一个多到一的映射，它是造成DCT编解码信息损失的根源。

根据人眼的视觉特性，对不同的变换系数设置不同的量化步长。

$$Q(u, v) = \text{round}\left[\frac{F(u, v)}{S(u, v)}\right]$$



3.4.2 基于DCT的图像编码

JPEG标准中亮度DCT系数的量化步长

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



3.4.2 基于DCT的图像编码

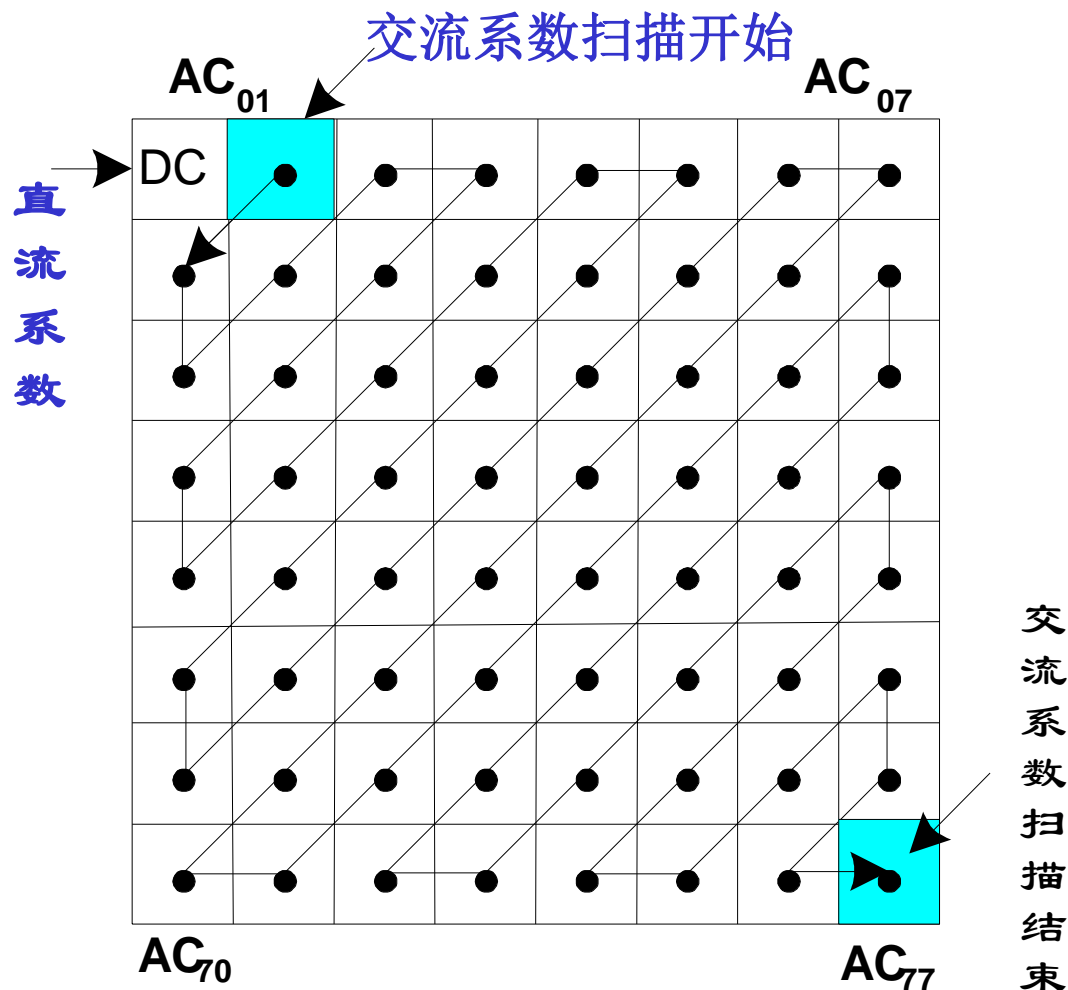
JPEG标准中色度DCT系数的量化步长

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

3.4.2 基于DCT的图像编码

变换系数熵编码

❧ Zig-Zag（或称“Z”字形，“之”字形）扫描





3.4.2 基于DCT的图像编码

变换系数熵编码

- ◆ **直流分量 (DC)** : 相邻图像子块的直流分量 (图像子块的平均样值) 也存在着相关性, 所以对DC的量化系数用**DPCM**编码较合适, 即对当前块和前一块的DC系数的差值进行编码。
- ◆ **交流分量 (AC)** : 把数值为0的连续长度 (即0游长) 和非0值结合起来构成一个**事件 (Run, Level)** , 然后再对事件 (Run, Level) 进行熵编码。

3.4.2 基于DCT的图像编码

8×8亮度子块

$$f(x, y)_{8 \times 8} = \begin{bmatrix} 78 & 75 & 79 & 82 & 82 & 86 & 94 & 94 \\ 76 & 78 & 76 & 82 & 83 & 86 & 85 & 94 \\ 72 & 75 & 67 & 78 & 80 & 78 & 74 & 82 \\ 74 & 76 & 75 & 75 & 86 & 80 & 81 & 79 \\ 73 & 70 & 75 & 67 & 78 & 78 & 79 & 85 \\ 69 & 63 & 68 & 69 & 75 & 78 & 82 & 80 \\ 76 & 76 & 71 & 71 & 67 & 79 & 80 & 83 \\ 72 & 77 & 78 & 69 & 75 & 75 & 78 & 78 \end{bmatrix}$$

第一步：DCT变换

$$\text{DCT} \quad F(u, v)_{8 \times 8} = \begin{bmatrix} 619 & -29 & 8 & 2 & 1 & -3 & 0 & 1 \\ 22 & -6 & -4 & 0 & 7 & 0 & -2 & -3 \\ 11 & 0 & 5 & -4 & -3 & 4 & 0 & -3 \\ 2 & -10 & 5 & 0 & 0 & 7 & 3 & 2 \\ 6 & 2 & -1 & -1 & -3 & 0 & 0 & 8 \\ 1 & 2 & 1 & 2 & 0 & 2 & -2 & -2 \\ -8 & -2 & -4 & 1 & 2 & 1 & -1 & 1 \\ -3 & 1 & 5 & -2 & 1 & -1 & 1 & -3 \end{bmatrix}$$



3.4.2 基于DCT的图像编码

第二步：量化。将DCT系数矩阵 $[F(u,v)]$ 中的每个元素与量化步长矩阵 $[S(u,v)]$ 中的对应元素相除后，进行四舍五入运算。

$$Q(u,v) = \text{round} \left[\frac{F(u,v)}{S(u,v)} \right]$$

$$\text{round} \left[\frac{8}{15} \right] = \text{round}[0.533] = 1$$

$$\text{round} \left[\frac{8}{16} \right] = \text{round}[0.5] = 1$$

$$\text{round} \left[\frac{7}{16} \right] = \text{round}[0.4375] = 0$$

$$Q(u,v)_{8 \times 8} = \begin{bmatrix} 39 & -3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



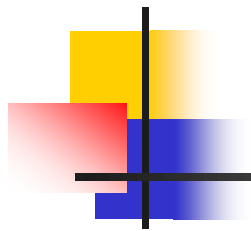
3.4.2 基于DCT的图像编码

第一步：恢复量化系数矩阵，将EOB后的元素自动补零。

第二步：反量化（IQ）

$$F'(u, v) = Q(u, v) \times S(u, v)$$

$$F'(u, v)_{8 \times 8} = \begin{bmatrix} 624 & -33 & 10 & 0 & 0 & 0 & 0 & 0 \\ 24 & -12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 14 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -17 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



第三步：DCT反变换（IDCT）

$$f'(x, y)_{8 \times 8} = \begin{bmatrix} 74 & 75 & 77 & 80 & 85 & 91 & 95 & 98 \\ 77 & 77 & 78 & 79 & 82 & 86 & 89 & 91 \\ 78 & 77 & 77 & 77 & 78 & 81 & 83 & 84 \\ 74 & 74 & 74 & 74 & 75 & 78 & 81 & 82 \\ 69 & 69 & 70 & 72 & 75 & 78 & 82 & 84 \\ 68 & 68 & 69 & 71 & 75 & 79 & 82 & 84 \\ 73 & 73 & 72 & 73 & 75 & 77 & 80 & 81 \\ 78 & 77 & 76 & 75 & 74 & 75 & 76 & 77 \end{bmatrix}$$

$$f(x, y)_{8 \times 8} = \begin{bmatrix} 78 & 75 & 79 & 82 & 82 & 86 & 94 & 94 \\ 76 & 78 & 76 & 82 & 83 & 86 & 85 & 94 \\ 72 & 75 & 67 & 78 & 80 & 78 & 74 & 82 \\ 74 & 76 & 75 & 75 & 86 & 80 & 81 & 79 \\ 73 & 70 & 75 & 67 & 78 & 78 & 79 & 85 \\ 69 & 63 & 68 & 69 & 75 & 78 & 82 & 80 \\ 76 & 76 & 71 & 71 & 67 & 79 & 80 & 83 \\ 72 & 77 & 78 & 69 & 75 & 75 & 78 & 78 \end{bmatrix}$$

重建后的信号与
原信号相差很小

主要原因是
量化所致。



Question?

