

# 新一代视频压缩编码标准

## ——H.264/AVC

毕厚杰主编

# 前 言

数字视频技术在通信和广播领域获得了日益广泛的应用，特别是 90 年代以来，随着 Internet 和移动通信的迅猛发展，视频信息和多媒体信息在 Internet 网络和移动网络中的处理和传输成为了当前我国信息化中的热点技术。

众所周知，视频信息具有一系列优点，如直观性、确切性、高效性、广泛性等等。但是视频信息量太大，要使视频得到有效的应用，必须首先解决视频压缩编码问题，其次解决压缩后视频质量保证的问题。这两者是相互矛盾的，是矛盾的俩个方面。我们的任务是既要有一定的压缩比，又要保证一定的视频质量。

为此，人们付出了巨大的辛勤的劳动，现已结出丰硕的成果。从 1984 年 CCITT 公布第一个视频编码国际标准以来，至今已有二十年了。ITU-T 等国际标准化组织陆续颁布了接近十个视频编码国际标准，大大推动了视频通信和数字电视广播的发展，这也是有目共睹的事实。但是严格地讲，这两大领域至今的发展仍不能令人满意，总起来讲，应用的范围不广，主要是视频压缩与质量之间的矛盾不能很好解决。例如，可视电话一直被认为是一种理想的通信设备，可近 30 年来至今未能普及，就是因为性能价格比不高。

2003 年 3 月，ITU-T/ISO 正式公布了 H.264 视频压缩标准，由于其相比以往标准的出色的性能，被人们称为新一代视频编码标准。具体讲，与 H.263 或 MPEG-4 相比，在同样质量下，其数码率能降低一半左右；或者说在同样码率下，其信噪比明显提高。这样一来，H.264 标准在国际上受到了广泛地重视和欢迎。在这样的背景下，我们编写了这一本书。

本书的特点是取材新颖、内容全面。它不仅重点论述了 H.264，而且首先介绍了数字视频和视频编码的基础知识，介绍了已有若干视频编码国际标准（特别是 MPEG-4），以便为进一步学习 H.264 打下良好的基础。

全书共 9 章，在 H.264 部分（第 6~9 章）详尽地论述了 H.264 特点、编码器原理、解码器原理、编解码器的实现。为了更好地理解 H.264 编解码原理及其实现，第 7 章详细介绍了 H.264 码流的句法和语义。最后对 H.264 视频编码传输的 QoS 进行了专门地论述。

本书可作为通信、广播电视专业高校本科生教材，可供该领域的硕士生、博士生深入研究用，也可供广大的从事视频技术、视频服务领域的技术人员参考用。

本书由毕厚杰教授主编，撰写第 1~4 章及第 6 章前 3 节。左雯撰写了第 5 章。马国强、徐苏珊撰写了第 7 章。焦良葆、王健撰写了第 9 章。其余部分由方晖、焦良葆、王健、马国强、左雯、李涛、徐苏珊、鹿宝生等人共同编写完成。全书由毕厚杰负责审稿，左雯负责校样。

由于时间仓促及水平有限，书中难免出现不当之处，恳请广大读者批评指正，以便再版时进一步修正。

毕厚杰  
2004.10.9

# 目 录

前 言 .....	2
第 1 章 绪 论 .....	9
1.1 信息化与视频通信 .....	9
1.1.1 什么是信息 .....	9
1.1.2 什么是信息化 .....	9
1.1.3 我国的信息化和视频通信 .....	9
1.2 视频信息和信号的特点 .....	10
1.2.1 直观性 .....	10
1.2.2 确切性 .....	10
1.2.3 高效性 .....	10
1.2.4 广泛性 .....	10
1.2.5 视频信号的高带宽性 .....	10
1.3 视频压缩编码要求和可能性 .....	11
1.3.1 视频压缩编码目标 .....	11
1.3.2 视频压缩的可能性 .....	11
1.4 视频压缩编码技术综述 .....	12
1.4.1 基本结构 .....	12
1.4.2 基于波形的编码 .....	12
1.4.3 基于内容的编码 .....	13
1.4.4 三维（立体）视频编码 .....	13
参考文献 .....	14
第 2 章 数字视频 .....	15
2.1 数字电视的基本概念 .....	15
2.1.1 数字电视的优越性 .....	15
2.1.2 数字电视的 PCM 原理 .....	16
2.2 数字电视信号 .....	18
2.2.1 电视信号的时间和空间取样 .....	18
2.2.2 彩色空间 .....	19
2.2.3 彩色电视取样格式 .....	19
2.2.4 数字电视信号的编码参数 .....	20
2.3 视频信号的预处理 .....	21
2.3.1 色彩插值（Color Interpolation） .....	21
2.3.2 色彩校正（Color Correction） .....	22
2.3.3 伽马校正（Gamma Correction） .....	23
2.3.4 图像增强（Image Enhancement） .....	24
2.3.5 白平衡（White Balance） .....	27
2.4 视频质量 .....	29
2.4.1 主观质量的评定 .....	29
2.4.2 客观质量的测量 .....	29
参考文献 .....	31

<b>第3章 视频压缩编码的基本原理</b>	<b>32</b>
3.1 预测编码	32
3.1.1 预测编码的基本概念	32
3.1.2 帧内预测编码	33
3.1.3 帧间预测编码	37
3.1.4 运动估计	41
3.2 变换编码	54
3.2.1 变换编码的基本概念	54
3.2.2 K-L 变换	54
3.2.3 离散余弦变换 DCT	55
3.2.4 锯齿形扫描和游程编码	57
3.3 变换编码与预测编码的比较	58
3.4 熵编码	59
3.4.1 变长编码	59
3.4.2 算术编码	59
参考文献	62
<b>第4章 视频编码标准简介</b>	<b>63</b>
4.1 视频编码发展简史	63
4.2 H.261 标准	63
4.2.1 图像格式	63
4.2.2 H.261 视频编解码器	64
4.3 H.263 标准	68
4.3.1 H.263 标准图像格式	68
4.3.2 H.263 视频信源编码算法	69
4.4 MPEG-1 标准	69
4.4.1 功能	69
4.4.2 图像类型和编码结构	69
4.5 MPEG-2	70
4.5.1 MPEG-2 编码复用系统	70
4.5.2 档次和级别	71
4.5.3 MPEG-2 视频编码器	72
4.6 JPEG 标准	73
参考文献	76
<b>第5章 MPEG-4 压缩编码标准</b>	<b>77</b>
5.1 MPEG4 标准概述	77
5.1.1 MPEG-4 标准特性	77
5.1.2 AVO 及数据结构	77
5.2 MPEG-4 标准构成	78
5.2.1 系统	78
5.2.2 音频	80
5.2.3 视频	80
5.2.4 网格动画	84

5.2.5 其余.....	85
5.3 MPEG-4 编码技术.....	85
5.3.1 形状编码.....	85
5.3.2 可扩展性编码.....	87
5.3.3 sprite 编码.....	90
5.3.4 视频系统合成.....	91
5.4 MPEG-4 档次和级.....	93
参考文献.....	96
<b>第 6 章 H.264/AVC 编码器原理.....</b>	<b>97</b>
6.1 H.264/AVC 的应用.....	97
6.2 H.264/AVC 编解码器.....	98
6.2.1 H.264 编解码器特点.....	98
6.2.2 H.264 编码器.....	98
6.2.3 H.264 解码器.....	99
6.3 H.264/AVC 的结构.....	99
6.3.1 名词解释.....	99
6.3.2 档次和级.....	99
6.3.3 编码数据格式.....	100
6.3.4 参数图像.....	102
6.3.5 片和片组.....	102
6.4 帧内预测.....	104
6.4.1 4×4 亮度预测模式.....	105
6.4.2 16×16 亮度预测模式.....	107
6.4.3 8×8 色度块预测模式.....	109
6.4.4 信号化帧内预测模式.....	109
6.5 帧间预测.....	110
6.5.1 树状结构运动补偿.....	110
6.5.2 运动矢量.....	111
6.5.3 MV 预测.....	113
6.5.4 B 片预测.....	114
6.5.5 加权预测.....	117
6.6 H.264 的 SP/SI 帧技术（SP 片或 SI 宏块的 P 宏块）.....	117
6.6.1 SP/SI 帧的应用.....	118
6.6.2 SP/SI 帧的基本原理.....	120
6.6.3 实验结果和性能分析.....	123
6.7 整数变换与量化.....	124
6.7.1 整数变换.....	125
6.7.2 量化.....	129
6.7.3 DCT 直流系数的变换量化.....	131
6.8 CAVLC(基于上下文自适应的可变长编码).....	132
6.8.1 熵编码的基本原理.....	132
6.8.2 CAVLC 的基本原理.....	133
6.8.3 CAVLC 的上下文模型.....	133
6.8.4 CAVLC 的编码过程.....	133

6.8.5 CAVLC 解码过程.....	135
6.8.6 CAVLC 编解码过程实例.....	138
6.8.7 CAVLC 与 UVLC 比较.....	139
6.9 CABAC(基于上下文的自适应二进制算术熵编码).....	141
6.9.1 自适应算术编码.....	141
6.9.2 上下文模型.....	144
6.9.3 对输入流预编码.....	146
6.9.4 初始化.....	147
6.9.5 结论.....	147
6.10 码率控制.....	148
6.10.1 基于 Lagrangian 优化算法的 H.264 编码控制模型.....	148
6.10.2 实验结果和性能分析.....	151
6.11 去方块滤波.....	153
6.11.1 去方块滤波基本概念.....	154
6.11.2 边界分析.....	155
6.11.3 滤波过程.....	157
6.12 其余特征.....	160
6.12.1 参考图像管理.....	160
6.12.2 重排序.....	160
6.12.3 隔行视频.....	161
6.12.4 数据分割片.....	162
6.12.5 H.264 传输.....	162
参考文献.....	164
<b>第 7 章 H.264 的句法和语义.....</b>	<b>166</b>
7.1 句法.....	166
7.1.1 句法元素的分层结构.....	166
7.1.2 句法的表示方法.....	169
7.2 句法表.....	170
7.3 语义.....	189
7.3.1 NAL 层语义.....	190
7.3.2 序列参数集语义.....	192
7.3.3 图像参数集语义.....	194
7.3.4 片头语义.....	196
7.3.5 参考图像序列重排序的语义.....	201
7.3.6 加权预测的语义.....	202
7.3.7 参考图像序列标记 (marking)操作的语义.....	203
7.3.8 片数据的语义.....	204
7.3.9 宏块层的语义.....	205
7.3.10 宏块预测的语义.....	210
7.3.11 子宏块预测的语义.....	211
7.3.12 用 CAVLC 方式编码的残差数据的语义.....	213
7.3.13 用 CABAC 方式编码的残差数据的语义.....	213
7.4 总结.....	213
参考文献.....	214

<b>第 8 章 H.264/AVC 解码器的原理和实现</b>	<b>215</b>
8.1 解码器原理	215
8.2 NAL 单元	216
8.2.1 NAL 单元结构	216
8.2.2 NAL 单元解码过程	216
8.3 图像序列号 (PICTURE ORDER COUNT) 的计算	217
8.3.1 图像序列号 (POC)	217
8.3.2 POC 类型为 0 的 POC 计算	219
8.3.3 POC 类型为 1 的 POC 计算	220
8.3.4 POC 类型为 2 的 POC 计算	221
8.4 宏块片组映射图的产生	221
8.5 片数据分割的解码	223
8.6 参考图像列表的初始化	224
8.6.1 图像序号的计算	224
8.6.2 参考图像列表的初始化	225
8.6.3 参考帧列表的重排序	228
8.7 解码的参考图像的标记过程	230
8.7.1 frame_num 不连续的解码过程	231
8.7.2 参考图像滑窗标记过程	231
8.7.3 参考图像的自适应内存控制标记过程	231
8.8 帧内预测	233
8.8.1 4x4 亮度块预测方式的提取	234
8.8.2 4x4 亮度块的帧内预测编码方式	235
8.8.3 16x16 亮度块的帧内预测方式	240
8.8.4 8x8 色度块的帧内预测方式	243
8.9 帧间预测解码处理	245
8.9.1 MV 分量及参考索引获取	246
8.9.2 帧间预测像素解码处理	248
8.10 变换系数解码	254
8.10.1 变换系数逆扫描过程	255
8.10.2 DCT 变换系数中直流系数的逆变换量化	256
8.10.3 残差变换系数的反量化	258
8.10.4 残差变换系数的逆 DCT 变换	258
8.10.5 去方块滤波前的图像恢复与重建	258
8.11 SP 片中的 P 宏块和 SI 片中的 SI 宏块的解码过程	259
8.11.1 主 SP 片中 P 宏块的解码过程	259
8.11.2 辅 SP/SI 片的解码过程	261
参考文献	263
<b>第 9 章 H.264 视频编码传输的 QoS</b>	<b>264</b>
9.1 互联网视频传输 QoS	264
9.1.1 错误恢复在视频通信中的重要性和实现途径	264
9.1.2 基于块的混合视频编码框架	265
9.1.3 视频通信中提高 QoS 的抗误码和错误恢复技术	266

9.2 无线网视频传输 QoS.....	271
9.2.1 无线视频通信系统.....	271
9.2.2 无线信道编码和错误控制.....	272
9.2.3 无线视频通信的应用.....	276
9.2.4 H.264 无线通信中传输结构.....	277
9.2.5 无线视频传输的鲁棒性研究.....	278
9.3 H.264 视频编解码标准的错误恢复.....	279
9.3.1 H.264 的视频编码层的错误恢复.....	279
9.3.1 H.264 的网络提取层的错误恢复.....	281
参考文献.....	284
<b>术语及英文解释.....</b>	<b>286</b>
<b>附录一 CAVLC 相关码表.....</b>	<b>297</b>
<b>附录二 CABAC 相关码表.....</b>	<b>304</b>
<b>附录三 H. 264 档次和级.....</b>	<b>318</b>



# 第1章 绪论

## 1.1 信息化与视频通信

本书讨论视频编码之前，简要介绍信息化问题，即研讨当前信息社会背景，然后讨论信息化与视频通信的关系。

### 1.1.1 什么是信息

众所周知，人类社会的三大支柱是物质、能量和信息。具体而言，农业现代化的支持是物质；工业现代化的支柱是能量；而信息化的支柱是信息。

广义地讲，信息就是客观世界的描述和分析，它无所不在，无时不在，具有广泛性和通用性，这是信息的一个性质；信息没有重量，没有长度，具有抽象性，这是信息的另一个特性，但它确实存在。

信息的第三个特性是无限性。比如，关于物质的信息，物质具有无限的不可分性，物质由分子组成，分子由原子组成，下面还有中子、质子、电子、中微子等等；关于通信网络的信息，为了增加通信容量，最初一对电话线只能通一对电话用户，后来利用 N-ISDN 技术，可在一对电话线上同时通两对电话，以后又发展 B-ISDN、ATM、IP、MPLS，直到今天，通信网络技术仍在不断地发展，应该说，它具有无限性。

总之，信息具有通用性、抽象性、无限性三个特征，其中无限性最重要。

从以上描述可知，信息是很有用的，它是客观世界中最本质的客观规律描述和分析，是人类社会可用的重要资源。信息资源如果能被充分开发和利用，人类社会的物质和精神文明水准将大大提高。

地球上的物质资源是有限的。石油、煤等在一定期限内总会开采完，但信息资源是无限的，对客观规律性的探讨是无尽的。而信息是要通过不断研究、不断分析，通过科学研究、反复实践才能掌握的。

### 1.1.2 什么是信息化

信息化是这样一个过程：“在现代信息技术广为普及的基础上，通过开发和利用信息资源，各种社会和经济活动的功能和效益显著提高了。人类社会的物质和精神文明达到了一个新的水平。”

可见，信息化的目的是提高人类的社会和经济效益。而实现信息化的关键在于开发和利用各种信息资源。

信息化的实现依赖于图 1.1 所示的信息系统构建。

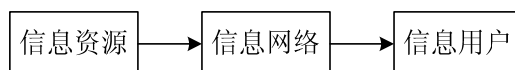


图 1.1 信息系统

信息化的实现首先要开发和利用各种信息资源，其次要有一个信息网络（如宽带通信网），通过该网络将大量信息传送到信息用户。

### 1.1.3 我国的信息化和视频通信

我国已成为规模上世界第一的通信大国。电话（包括固定和移动电话）的普及率已达到 30%。宽带网络正在不断建设中，8 纵 8 横的光纤构成了我国骨干网。以 MSTP（多业务传输平台）技术为主的城域网正在大力建设，以 ADSL 为主的宽带接入网也迅速发展。应该指出，我国信息化的瓶颈在信息资源的开发和利用。

经常有这样的疑问：有了宽带网络，传送什么内容呢？实际上，世界上的信息资源是十分丰富的，问题在于人们没有努力开发或者充分利用。

信息资源中，视频信息的开发、利用更具有重要的理论意义和应用价值。

90年代初以来，我国的会议电视获得了巨大发展，短短几年，从最初的一个中央到各省的会议电视骨干网，逐步发展为铁道、电力、石油、公安多系统、各省以至各地县约数千个的会议电视网。

90年代 Internet 的迅猛发展，导致 IP 技术应用普及到各个方面，网上教育、远程医疗、电子商务、电子政务、电子游戏、网上证券等等如雨后春笋。一句话，IP 视频通信已十分引人注目。

通信业务已从电话、传真、电报发展为可视电话、会议电视、视频点播等，即由音频为主发展到以视频为主的多媒体通信。

为了更深入的理解视频通信，有必要对它的特点进一步探讨。

## 1.2 视频信息和信号的特点

### 1.2.1 直观性

利用人的视觉获取的信息称为视频信息，它具有直观性的特点。话音信息则利用人的听觉获取。两者相比，视频信息给人印象深刻、具体，话音信息则相对较浅。从交流信息的客观效果讲，视频信息的效果更好。

### 1.2.2 确切性

视频信息直观具体，不易和其他内容相混淆，保证信息的准确性。而话音则会由于地方口音的不同产生歧义，导致不必要的损失。

### 1.2.3 高效性

由于利用视觉，人们可以并行地观察一幅图像的各个像素，因而获取视频信息的效率比音频信息高的多。例如，人们通过一幅电机构建的图，很快搞清楚定子、转子及其相关位置，从而很快弄清电机的结构及其原理；如果人们没有这样的图，只是一味的讲课，从这样的音频信息中反复讲解电机结构，也可能仍搞不清楚，其接受的效率要低的多。

### 1.2.4 广泛性

据统计，人们每天视觉获取的视觉信息约占外界信息总量的 70% 左右，即人们每天获得的信息大部分是视觉信息。

### 1.2.5 视频信号的高带宽性

视频信息的表示形式是视频信号，通常为视频的电信号，通过网络传送至终端用户，并在屏幕上显示。

视频信号所包含的信息量大，其内容可以是活动的，也可以是静止的；可以是彩色的，也可以是黑白的；有时变化多、细节多，有时十分平坦。一般而言，视频信号信息量大，传输网络所需要的带宽相对较宽。例如，一路可视电话或会议电视信号，由于其活动内容较少，所需带宽较窄，但要达到良好质量，不压缩约需若干 Mbps，压缩后需要 384Kbps；又如，一路高清晰度电视信号（HDTV），由于其信息量相当巨大，不压缩需 1Gbps，利用 MPEG-2 压缩后，尚需 20Mbps。可见，视频信息虽然具有直观性、确定性、高效性等优越性能，但要传送包含视频信息的信号却需要较高的网络带宽。这就是为获得视频信息所需付出的代价。

## 1.3 视频压缩编码要求和可能性

### 1.3.1 视频压缩编码目标

如上所述，视频信号由于信息量大，传输网络带宽要求高，就像一辆庞大的货车只有在宽阔的马路上才能行驶一样。于是出现一个问题：能否将视频信号在传送前先进行压缩编码，即进行视频源压缩编码，然后在网络上进行传送，以便节省传送带宽和存储空间。这里有两个要求：

- 1) 必须压缩在一定的带宽内，即视频编码器应具有足够的压缩比；
- 2) 视频信号压缩之后，应保持一定的视频质量。这个视频质量有两个标准：一个为主观质量，由人从视觉上进行评定；一个为客观质量，通常用信噪比（S/N）表示。

如果不问质量，一味地压缩，虽然压缩比很高，但压缩后严重失真，显然达不到要求；反之，如只讲质量，压缩比太小，也不符合要求。

当然，在以上两个要求下，视频编码器的实现应力求简单、易实现、成本低、可靠性高，这也是基本的要求。

### 1.3.2 视频压缩的可能性

#### 1.3.2.1 预测编码

前面讨论了视频信息的优越性，视频信号压缩的必要性，也提出了视频压缩的目标（要求），但实现这些目标的可能性如何？

众所周知，一幅图像由许多个所谓像素的点组成，如图 1.2 中的“O”表示一个像素，大量的统计表明，同一幅图像中像素之间具有较强的相关性，两个像素之间的距离越短，则其相关性越强，通俗地讲，即两个像素的值越接近。换言之，两个相邻像素的值发生突变的概率极小，“相等、相似或缓变”的概率则极大。

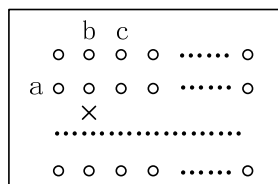


图 1.2 像素间相关性解释

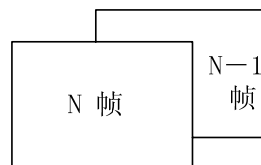


图 1.3 帧间相关性解释

于是，人们可利用这种像素间的相关性进行压缩编码。例如当前像素 X（设为立即传送的像素）可用前一个像素 a 或 b、c，或三者的线性加权来预测。这些 a，b，c 被称为参考像素。在实际传送时，把实际像素 X（当前值）和参考像素（预测值）相减，简单起见传送  $X - a$ ，到了接收端再把  $(X - a) + a = X$ ，由于 a 是已传送的（在接收端被存储），于是得到当前值。由于 X 与 a 相似， $(X - a)$  值很小，视频信号被压缩，这种压缩方式称为帧内预测编码。

不仅如此，还可利用图 1.3 所示的帧间相关性进行压缩编码。由于邻近帧之间的相关性一般比帧内像素间的相关性更强，压缩比也更大。

由此可见，利用像素之间（帧内）的相关性和帧间的相关性，即找到相应的参考像素或参考帧作为预测值，可以实现视频压缩编码。

#### 1.3.2.2 变换编码

大量统计表明，视频信号中包含着能量上占大部分的直流和低频成分，即图像的平坦部分，也有少量的高频成分，即图像的细节。因此，可以用另一种方法进行视频编码，将图像经过某种数学变换后，得到变换域中的图像（如图 1.4 所示），其中 u，v 分别是空间频率坐标。如图 1.4 所示，用

“o”表示的低频和直流占图像能量中的大部分，而高频成分（用“×”表示）则是少量的，于是可用较少的码表示直流低频以及高频，而“O”则不必用码，结果完成了压缩编码。

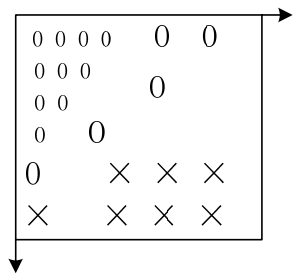


图 1.4 变换域图像

## 1.4 视频压缩编码技术综述

### 1.4.1 基本结构

视频编码系统的基本结构如图 1.5 所示。

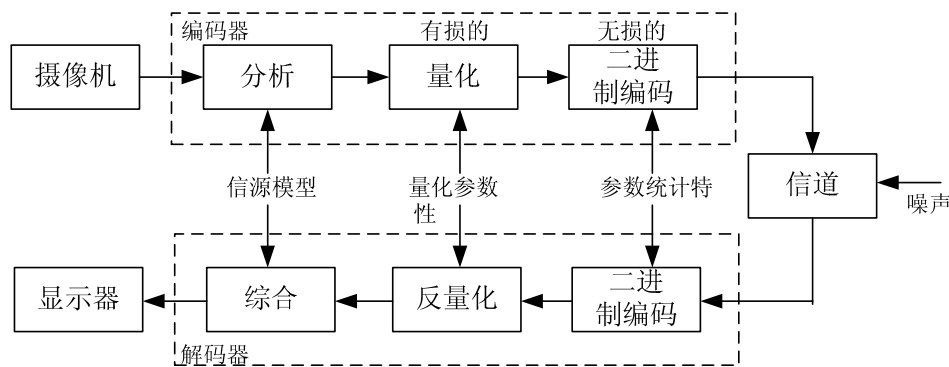


图 1.5 视频编码系统

由图 1.5 可见，视频编码方法与可采用的信源模型有关。如果采用“一幅图像由许多像素构成”的信源模型，这种信源模型的参数就是每个像素的亮度和色度的幅度值。对这些参数进行压缩编码技术称为基于波形的编码。如果采用一个分量有几个物体构成的信源模型，这种信源模型的参数就是各个物体的形状、纹理和运动。对这些参数进行压缩编码的技术被称为基于内容的编码。

由此可见，根据采用信源模型，视频编码可以分为两大类，基于波形的编码和基于内容的编码。它们利用不同的压缩编码方法，得到相应的量化前的参数；再对这些参数进行量化，用二进制码表示其量化值；最后，进行无损熵编码进一步提高码率。解码则为编码的逆过程。

### 1.4.2 基于波形的编码

如上所述，利用像素间的空间相关性和帧间的时间相关性，采用预测编码和变换编码技术可大大减少视频信号的相关性，从而显著降低视频序列的码率，实现压缩编码的目标。

基于波形的编码采用了把预测编码和变换编码组合起来的基于块的混合编码方法。

为了减少编码的复杂性，使视频编码操作易于执行，采用混合编码方法时，首先把一幅图像分成固定大小的块，例如块 8×8（即每块 8 行，每行 8 个像素）、块 16×16（每块 16 行，每行 16 个像素）等等，然后对块进行压缩编码处理。

自 1989 年 ITU-T 发布第一个数字视频编码标准——H.261 以来，已陆续发布了 H.263 等视频编码标准及 H.320、H.323 等多媒体终端标准。ISO 下属的运动图像专家组（MPEG）定义了 MPEG-1、MPEG-2、MPEG-4 等娱乐和数字电视压缩编码国际标准。

2003 年 3 月份，ITU-T 颁布了 H.264 视频编码标准。它不仅使视频压缩比较以往标准有明显提高，而且具有良好的网络亲和性，特别是对 IP 互联网、无线移动网等易误码、易阻塞、QoS 不易保证的网络视频传输性能有明显的改善。本书的主要内容即 H.264 视频编码。

所有这些视频编码都采用了基于块的混合编码法，都属于基于波形的编码。

### 1.4.3 基于内容的编码

如上所述，基于块的编码易于操作，但由于人为地把一幅图像划分成许多固定大小的块，当包含边界的块属于不同物体时，它们分别具有不同的运动，便不能用同一个运动矢量表示该边界块的运动状态。如果强制划分成固定大小的块，这种边界块必然会产生高的预测误差和失真，严重影响了压缩编码信号的质量。

于是产生了基于内容的编码技术。这时先把视频帧分成对应于不同物体的区域，然后对其编码。具体说来，即对不同物体的形状、运动和纹理进行编码。在最简单情况下，利用二维轮廓描述物体的形状；利用运动矢量描述其运动状态；而纹理则用颜色的波形进行描述。

当视频序列中的物体种类已知时，可采用基于知识或基于模型的编码。例如，对人的脸部，已开发了一些预定义的线框对脸的特征进行编码，这时编码效率很高，只需少数比特就能描述其特征。

对于人脸的表情（如生气、高兴等），可能的行为可用语义编码，由于物体可能的行为数目非常小，可获得非常高的编码效率。

**MPEG-4 采用的编码方法就既基于块的混合编码，又有基于内容的编码方法。**

### 1.4.4 三维（立体）视频编码

立体视频编码是视频编码的发展方向之一，其平面信息外增加了深度信息，数据量非常庞大。

立体视频编码也有两种类型的方法：第一种是基于波形的，它组合运动补偿预测和位差补偿预测。所谓位差估计即在两幅不同图像中寻找对应的点。它对预测残差图像、位差和运动矢量进行编码；第二种是基于物体的，它直接对成像物体的三维结构和运动进行编码。当物体结构简单时，可获得非常高的压缩比。其编码结构如图 1.6 所示。

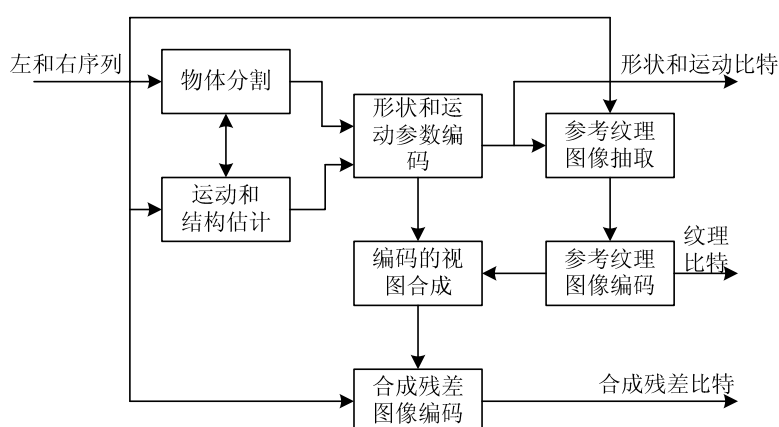


图 1.6 三维物体序列编码系统

## 参考文献

- 1 Yao Wang, Jorn Ostermann, Ya-Qin Zhang, Video Processing and Communication, Pearson Education, 2002.
- 2 毕厚杰, 多媒体信息的传输与处理, 人民邮电出版社, 1999
- 3 毕厚杰, 信息和信息化, 中国工程科学, 2003 第 5 卷第 5 期 92 页。

## 第 2 章 数字视频

视频压缩编码技术就是对数字的视频信号进行压缩和解压缩的过程。因此讨论视频压缩编码前，必须先了解数字视频信号的概念和构成。这是本章的主要内容。

什么是数字信号？它是自然电视景象的数字表示。具体说来，在本章中将依次讨论：（1）数字电视的概念；（2）彩色空间；（3）数字电视景象标准格式；（4）A/D 和 D/A 变换；（5）取样和亚取样；（6）量化；（7）数字视频的质量。

### 2.1 数字电视的基本概念

#### 2.1.1 数字电视的优越性

现在，模拟彩色电视已经相当普及，在一定程度上满足了人们的生活需求。但是，模拟电视缺陷日益暴露出来。为此，数字电视就应运而生，与模拟电视相比，具有许多突出的优点：

##### （1）失真小、噪声低、质量高

模拟电视信号在放大、处理、传输、存储过程中，难免会引入失真噪声，而且多种噪声与失真叠加到电视信号后，不易去除，且会随着处理次数和传输距离的增加不断累积，导致图像质量及信噪比的下降。

相反，数字电视信号没有上述的噪声累积效应。如图 2.1 所示，只要噪声电平不超过脉冲幅度的一半，就可用脉冲再生技术对其整形，并恢复成“0”或“1”两种电平，便不会引入噪声。这样说来，是否数字电视信号没有任何失真和噪声呢？从下面的叙述可知，它会引入“量化噪声”，这是因为信号的数字化必须要经过取样、量化、编码三个基本步骤，“量化”是不可缺少的。量化噪声不可避免，但可控制在相当低的电平一下。

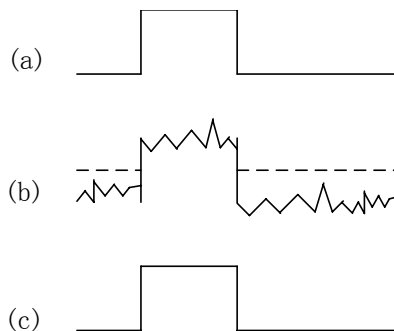


图 2.1 数字电视信号抗噪声能力

可见，数字电视的第一个突出优点是噪声低、失真小、视频质量好。

##### （2）易处理、易校正

模拟电视信号要进行压缩编码处理、加密处理、校正处理都不是一件容易的事情。

数字电视信号利用 VLSI 芯片进行压缩编码处理、加密处理、白平衡调整、 $\gamma$  校正、彩色校正和轮廓校正，相对来说容易得多。随着大规模集成电路设计和工艺的不断发展，现在利用专用芯片和通用 DSP 来实现以上处理的成本不断下降，这是视频数字压缩编码能取得不断发展的一个重要原因。

##### （3）容量大、节目多

同样带宽的模拟电路能容纳的数字电视节目比模拟电视节目多得多。例如，CATV 频道中，550MHz~750MHz 的 200MHz 带宽中，如果传送模拟电视，每个节目需 8MHz 带宽，充其量只能传送 25 套节目。如果传送数字电视节目，采用 64QAM 调制，其频谱利用率为 5.3，如每路节目用 MPEG-2 压缩为 4Mbps，实际只需  $4 \div 5.3 \approx 0.75$  MHz 带宽，于是在同样的 200MHz 带宽中可传送数字电视节目

数为  $200 \div 0.75 \approx 260$ ，约为模拟电视的 11 倍！

## 2.1.2 数字电视的 PCM 原理

将输入的模拟电视信号变成输出的数字电视信号需经过取样、量化、编码三个步骤，如图 2.2 所示，由 A/D 变换器完成这三个步骤。

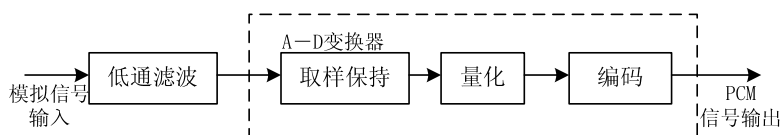


图 2.2 PCM 原理

### 2.1.2.1 取样

所谓取样，就是在时间轴上把连续变化的模拟信号变为离散量的过程。图 2.3 (a) 的  $u_a(t)$  在时间上是连续变化的，经取样后变换成图 2.3 (c) 的时间上离散的  $u_d(t)$  信号。

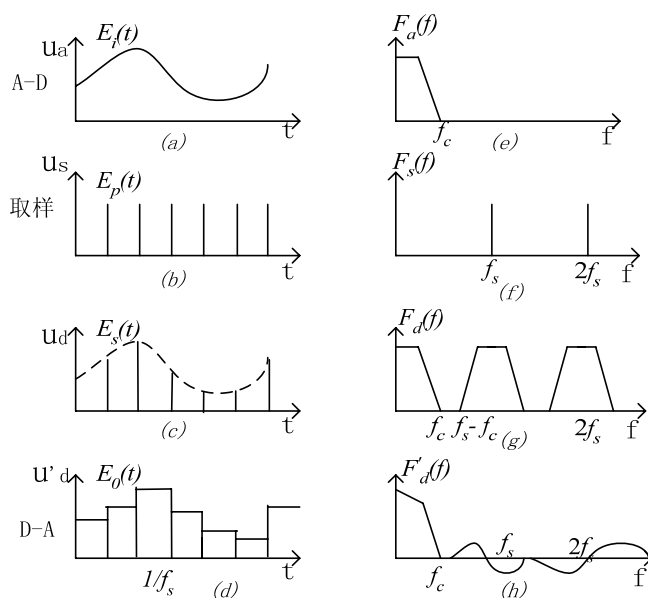


图 2.3 取样过程

根据取样定理：当输入的模拟信号上限频率为  $f_c$ ，只要取样脉冲  $u_s(t)$  的重复频率  $f_s$  不低于  $f_c$  的两倍，总可以无失真地由取样后的离散信号恢复出原来的模拟信号，即不失真输出条件为：

$$f_s \geq 2f_c \quad (2.1)$$

$u_d$  通过下式实现：

$$u_d = u_a \times u_s \quad (2.2)$$

图 2.3 (e)、(f) 分别是  $u_a(t)$  和  $u_s(t)$  的频谱  $F_a(f)$ 、 $F_s(f)$ 。于是图 2.3 (c)  $u_d(t)$  的频谱，即图 2.3 (g) 的  $F_d(f)$ ，可由  $F_a(f)$  和  $F_s(f)$  卷积得到：

$$F_d(f) = F_a(f) * F_s(f) \quad (2.3)$$

由图 2.3 (g) 可知，当  $2f_c \leq f_s$  或满足取样定理时，则可恢复出原始的模拟信号，否则会发生频谱重叠，即所谓的混叠效应，无法恢复出原始信号。

由于实际的低通滤波器（限制模拟信号的上限频率  $f_c$ ）滤波特性不可能做成理想的陡峭的截止特性，当低通滤波器的截止频率为  $f_c$  时，实际的取样频率  $f_s$  应取成：

$$f_s = (2.2 \sim 2.5) f_c \quad (2.4)$$

对于电视信号，经分析可知其信号能量主要集中在行频  $f_h$  及其多次谐波  $n f_h$  附近。而在  $f = (2n+1)$



$f_h/2$  附近, 信号能量很弱。当取样频率  $f_s$  取下式:

$$f_s = (2n+1) f_h/2 \quad (2.5)$$

$f_s < 2f_c$ , 即所谓的亚取样时, 发生频谱混叠, 但频谱以  $f_h/2$  的间隔交错开, 因此仍可通过设计得当的梳状滤波器将所需信号的频谱分离出来。

这种亚取样可显著压缩数字电视的数字码率。

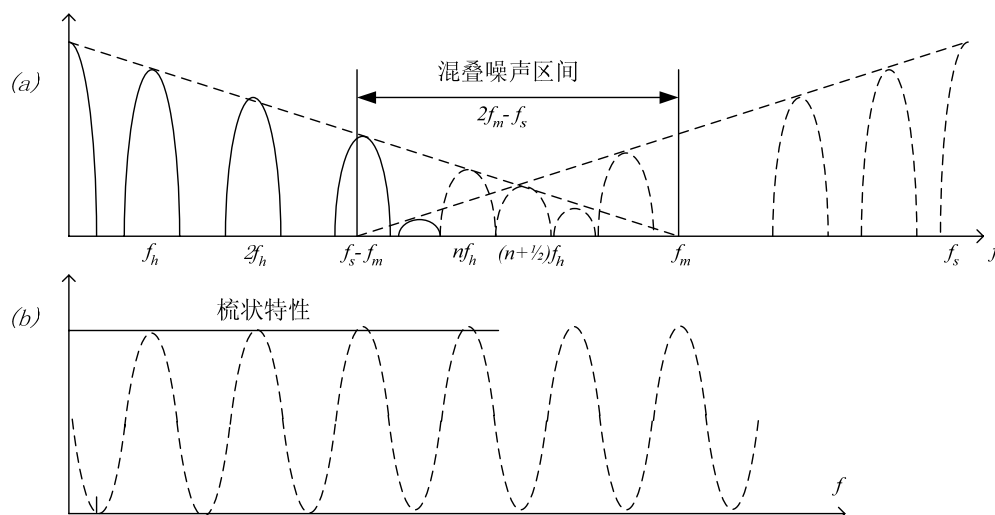


图 2.4 视频信号亚取样的频谱混叠

### 2.1.2.2 量化

取样后的脉冲信号在时间上是离散的, 但在幅值上空间上仍是连续的, 即其可能取的值有无限多个, 这就需要对它采用四舍五入的方法, 将其可能的幅值数由无限多个变为有限个值。这种将信号幅值由连续量变成离散量的过程称为量化。

图 2.5 为信号的量化过程。量化器的输入输出特性 (图 2.5 (a)) 具有阶梯形状, 图 2.5 (b) 为输入模拟信号, 图 2.5 (c) 为其相应的量化后的输出信号。由于采取四舍五入的方法, 输出信号不同于原模拟信号, 产生了失真, 即所谓的“量化噪声”。

如果模拟信号的动态范围 (最大值) 为  $A$ , 量化级数为  $M$ , 量化节距或量化步长为  $Q$ , 则

$$M = A/Q \quad (2.6)$$

这种量化称为均匀量化, 量化节距为恒定值  $Q$ 。

### 2.1.2.3 PCM 编码

对于量化后的信号, 通常用“0”和“1”表示, 即用二进制码表示。这时的编码称为脉冲编码调制——PCM 编码。模拟电视信号经取样、量化、编码 (PCM 编码) 后得到的二进制序列, 即数字电视信号。

每个取样信号用 8 位二进制码表示, 可能取的量化值为  $M = 2^8 = 256$ 。一般讲, 当用  $n$  位二进制码表示时, 有

$$M = 2^n \quad (2.7)$$

$n$  愈大, 则  $M$  愈大,  $Q$  愈小, 即量化噪声愈小, 数字信号愈接近模拟信号。

### 2.1.2.4 A/D 与 D/A 变换

上述取样、量化、编码过程均由 A/D 变换器完成。反之, 数字信号的解码、反量化、恢复成模拟信号的逆过程则由 D/A 变换器完成, 见图 2.5 所示。

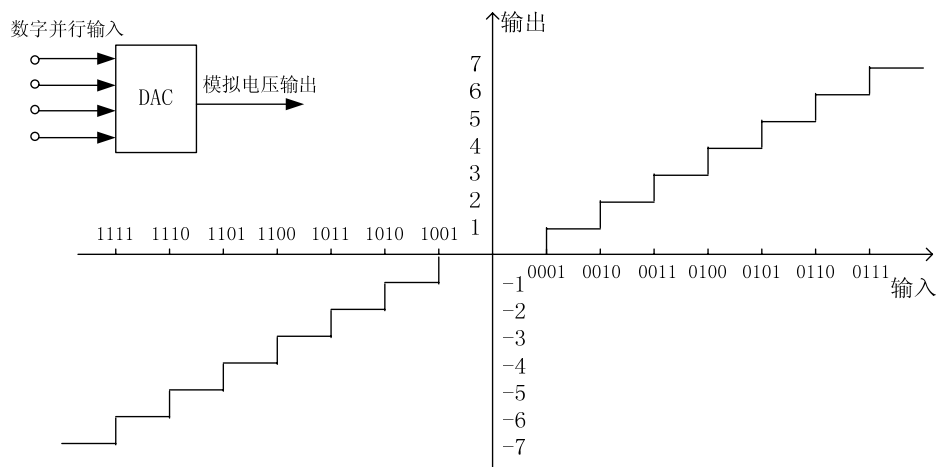


图 2.5 D-A 变换

## 2.2 数字电视信号

### 2.2.1 电视信号的时间和空间取样

#### 2.2.1.1 时间取样

电视信号的取样有两种：空间取样和时间取样。运动图像可由每秒若干帧静止图像构成，我国采用的 PAL 制彩色电视规定每秒 25 帧，美日等采用的 NTSC 制彩色电视则为每秒 30 帧。这种取样方式即时间取样。如果是会议电视、可视电话等运动量不大的视频信号，帧频也可取 15~20 帧/秒，但低于 15 帧/秒的视频质量不高。

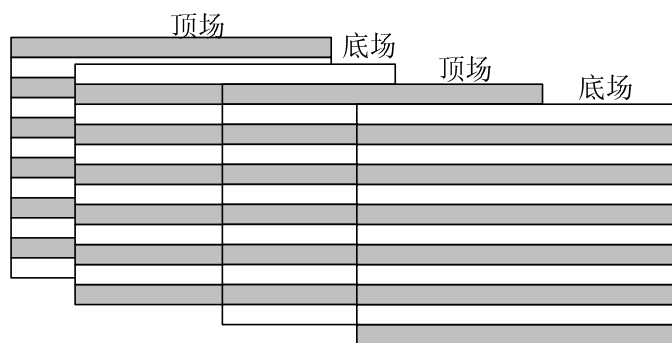


图 2.6 帧、场结构

隔行扫描帧图像由两场组成，每场由若干行组成，奇数行和偶数行各构成一场，它们分别为顶场和底场，如图 2.6 所示。帧场的邻近行相关性并不相同。帧的邻近行空间相关性强，时间相关性弱，因为某行的邻近行（下一行）要一场扫描完才能被扫描，在压缩静止图像或运动量不大的图像时采用帧编码方式。场的邻近行时间相关性强，空间相关性差，因为场的一行扫描完毕，接着对场中下一行扫描。因此对运动量大的图像常采用场编码方式。实际的视频图像有快有慢，有粗有细，应根据这个标准自适应选择帧/场编码方式。

#### 2.2.1.2 空间取样

在同一电视信号帧中，同一行由若干取样点构成，这些取样点称为像素，这种取样就属于空间

取样。其前提是假定一帧图像是静止的，每个像素点处于同一时刻及不同的空间位置上。

例如国际上标准电视格式为 720×576 像素，即每帧由 576 行，每行由 720 个像素构成；美国的 GA 制规定了两种扫描格式，即 720×1280 像素和 1080×1920 像素。

现存在三种彩色电视制式，不同的国家采用不同的制式，为了实现国际上的不同彩色电视制式国家之间视频通信，往往采用一种中间公共格式（CIF），如表 2.1 所示。

表 2.1 视频帧格式

格式	亮度清晰度
亚 QCIF	96×128
QCIF	144×176
CIF	288×352
4CIF	576×720

2.2.2 彩色空间

黑白图像的每个像素只需一个幅值表示其亮度即可；而彩色图像的每个像素至少需要 3 个值表示其亮度和色度。所谓色度空间即表示彩色图像的亮度与色度的方法。

2.2.2.1 RGB

众所周知，任何彩色图像可由不同比例的红色、绿色和蓝色组合而成，即三基色原理。这种表示彩色图像的方法即 RGB 彩色空间。

彩色显象管（CRT）和液晶显示器件（LCD）可显示彩色图像，彩色摄像机中的电荷耦合器件（CCD）等传感器可产生彩色电视图像，都是根据 RGB 原理获得的。

2.2.2.2 YCbCr(YUV)

人类视觉系统（HDV）对亮度比彩色更敏感，因此可以把亮度信息从彩色信息分离出来，并使之具有更高的清晰度，彩色信息的清晰度较低些，可显著压缩带宽，实现视频压缩的一部分，人的感觉却没有不同。

如果亮度分量用 Y 表示，色度用 Cb, Cr 表示，则由大量实验得出：

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ Cb &= 0.564(B - Y) \\ Cr &= 0.713(R - Y) \end{aligned} \tag{2.8}$$

反之，可由下式得到相应的 R、G、B：

$$\begin{aligned} R &= Y + 1.402Cr \\ G &= Y - 0.344Cb - 0.714Cr \\ B &= Y + 1.772Cb \end{aligned} \tag{2.9}$$

2.2.3 彩色电视取样格式

有三种不同的彩色电视取样格式，如图 2.7 所示。

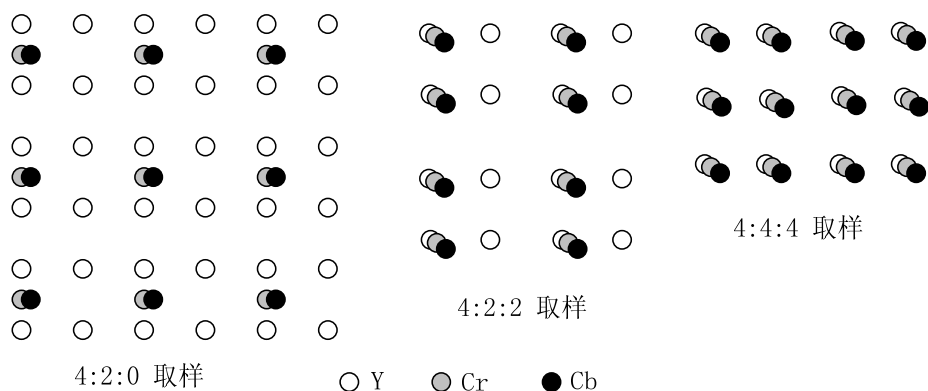


图 2.7 彩色电视取样格式（逐行）

(1) 4:4:4, Y, Cb 和 Cr 具有同样的水平和垂直清晰度, 在每一像素位置, 都有 Y, Cb 和 Cr 分量, 即不论水平方向还是垂直方向, 每 4 个亮度像素相应的有 4 个 Cb 和 4 个 Cr 色度像素。

(2) 4:2:2, 这时彩色分量和亮度分量具有同样的垂直清晰度, 但水平清晰度彩色分量是亮度分量的一半。水平方向上, 每 4 个亮度像素具有 2 个 Cb 和 2 个 Cr。在 CCIR601 标准中, 这是分量彩色电视的标准格式。

(3) 4:2:0, 在水平和垂直清晰度方面, Cb 和 Cr 都是 Y 的一半。这个名词是历史上一直称呼的, 似乎不合乎逻辑。

4:2:0 的彩电取样格式广泛应用于数字电视、会议电视、DVD 等。因为三种格式中, 4:2:0 的彩色分量最少, 对人彩色感觉而言与其它两种类似, 最适合数字压缩。

## 2.2.4 数字电视信号的编码参数

现在介绍电视信号的量化值和取样频率值, 即未压缩前数字信号的编码参数。

### 2.2.4.1 量化值 $Q_p$

量化值 (量化节距) 取得太大, 视频图像显得粗糙; 取得太小, 视频图像质量好, 但带宽浪费过大。一般认为, 每个取样值采用 8 个比特表示, 即 256 个灰度级, 是比较合理的。在会议电视的视频通信中, 随着网络带宽的变化,  $Q_p$  可进行自动调整。

### 2.2.4.2 取样频率

CCIR601 建议的电视国际标准为: 对每幅画面 625/50 (625 行, 每秒 50 场) 的电视系统和 525/60 (525 行, 每秒 60 场) 的电视系统取样频率都为:

$$f_s = 13.5\text{MHz} \quad (\text{亮度信号, 即 Y 信号})$$

$$f_s = 6.75\text{MHz} \quad (\text{色差信号, 即 Cb、Cr 信号})$$

彩色电视采用 4:2:2 格式时 (垂直方向 Cb、Cr 和 Y 具有同等清晰度, 水平方向 Cb、Cr 只是 Y 一半), Y 和 Cb、Cr 取样频率如上, 则电视信号总数码率为:

$$13.5 \times 8 + 2 \times 6.75 \times 8 = 216\text{Mbps}$$

当会议电视采用 CIF 格式时, 帧频为 25 帧/秒, 总码率为:

$$352 \times 288 \times 25 \times 8 = 20.28\text{Mbps}$$

对于高级窄屏幕的 HDTV (1250×1440), 取 4:2:0 格式, 亮度  $f_s = 54\text{MHz}$ , 色度  $f_s = 27\text{MHz}$ , 总码率为:

$$54 \times 8 + 27 \times 8 \div 4 = 486\text{Mbps}$$

对于高级宽屏幕的 HDTV (1250×1920), 取 4:2:0 格式, 亮度  $f_s = 72\text{MHz}$ , 色度  $f_s = 36\text{MHz}$ , 总码率为:

$$72 \times 8 + 36 \times 8 \div 4 = 648 \text{Mbps}$$

不论何种数字电视信号，这些值 20.28Mbps、216Mbps、486Mbps、648Mbps 都是没有经过压缩的码率。如果直接在现有的信道中传输，都需要相当的带宽，因此需对这些数字视频信号进行压缩编码。

## 2.3 视频信号的预处理

视频处理和通信系统所要处理的信息是十分庞大的视频图像数据，对于处理的速度和精度都有相当高的要求，系统的应用也是非常广泛。例如，在个人移动通信、远程医疗诊断设备、智能楼宇、联网交通监控、自然灾害预测以及国防建设等领域获得了惊人的成果。目前的视频处理和通信系统有各种各样的结构，但不论结构复杂还是简单，一个基本的视频处理和通信系统大致可如图 2.8 所示，主要包括采集、预处理、视频编码、通信、图像处理以及显示等几个方面。

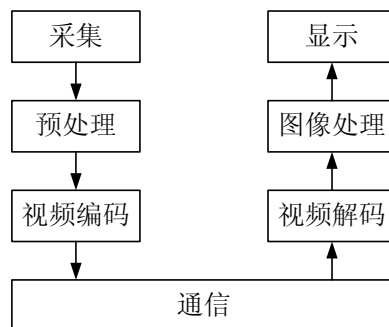


图 2.8 视频处理和通信系统

图像采集的功能由图像传感器实现，目前图像传感器主要有电荷耦合器件（CCD, charge coupled devices）和 CMOS 传感器，前者技术发展成熟，具有高解析度、低噪声、动态范围大等优点，在高端产品中得到广泛应用，后者随着半导体技术的发展，以其低成本、高的集成度、低功耗等占领了低端市场，且随着技术的不断发展，CMOS 图像传感器的一些参数性能指标已达到或超过 CCD。但不论是 CCD 还是 CMOS 传感器在将实际景物转换为图像信号时总会引入各种噪声和畸变失真，因此一般需要对图像传感器的图像进行预处理，包括伽马校正、图像插值、图像校正、白平衡、图像增强以及增益控制等技术，一方面改善图像的质量，另一方面，可使得图像有利于视频编码的处理。至于视频图像编码和通信在后续章节将进行重点论述。

### 2.3.1 色彩插值（Color Interpolation）

不论是 CCD 还是 CMOS 图像传感器，但为了简化工艺和降低成本，一个像素点往往只能给出记录从纯白到纯黑的系列色调，因而只能给出单色的色调值，不能同时给出 RGB 三组数据。因此，对于彩色的图像值的获取，这就需要借助色彩滤镜阵列（CFA, Color Filter Array），即图像传感器的像素表面覆盖一个多色的滤镜阵列。通过应用不同的色彩滤镜阵列，可以获得不同的图像输出阵列，其中，最常见的一种滤镜阵列的图像传感器获得的是一幅如图 2.9 所示的马赛克的图像阵列，即 Bayer 模型。

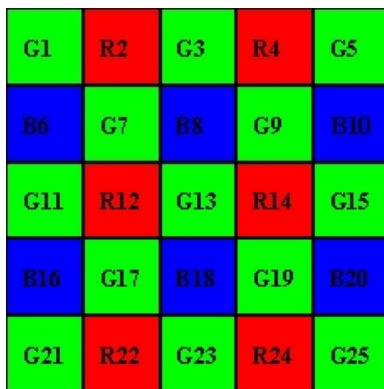


图 2.9 Bayer 图像阵列

显然，这种图像阵列中，每个像素值只有一个颜色的色调值，另外两个颜色的色调必须利用相邻像素之间的相关性，通过数据计算获得，这些方法通常就被称为色彩插值。这里就以 Bayer 图像阵列为基础，描述色彩的插值算法。文献[2]给出了多种色彩插值算法的实现和性能比较，例如，相邻像素复制法、双线性插值算法以及双三次多项式插值等等。这里仅就双线性插值算法为例做个简单的描述。显然，色彩的插值涉及 RGB 三色的处理。具体如下：

- 红色/蓝色点处的像素绿色分量

插值等于其相邻的四个像素点的绿色分量平均值。例如， $G8 = (G3 + G7 + G9 + G13) / 4$ 。

- 在绿色点处的红色/蓝色分量的插值

这分两种情况：一种情况是如果存在相邻的两个像素的红色/蓝色分量，就取红色/绿色分量的均值，例如  $B7 = (B6 + B8) / 2$ ， $R7 = (R2 + R12) / 2$ 。另一个情况是周围没有相邻的红色/蓝色分量，就取对象线方形的四角像素点处红色/蓝色分量的均值， $R8 = (R2 + R4 + R12 + R14) / 4$ ， $B12 = (B6 + B8 + B16 + B18) / 4$ 。

### 2.3.2 色彩校正（Color Correction）

上一节通过插值已经得到了 RGB 三元色齐全的图像（R、G、B）了，但传感器响应的这个图像与真实场景之间仍存在差异。这存在多方面的原因，涉及图像传感器中光学器件（棱镜）的光谱特性、场景的光源光照条件（诸如白光、荧光或者钨光）以及色彩滤镜的光谱特性等。图 2.10 给出了配置 RGB 色彩滤镜阵列的 CMOS 图像传感器的光谱响应曲线。

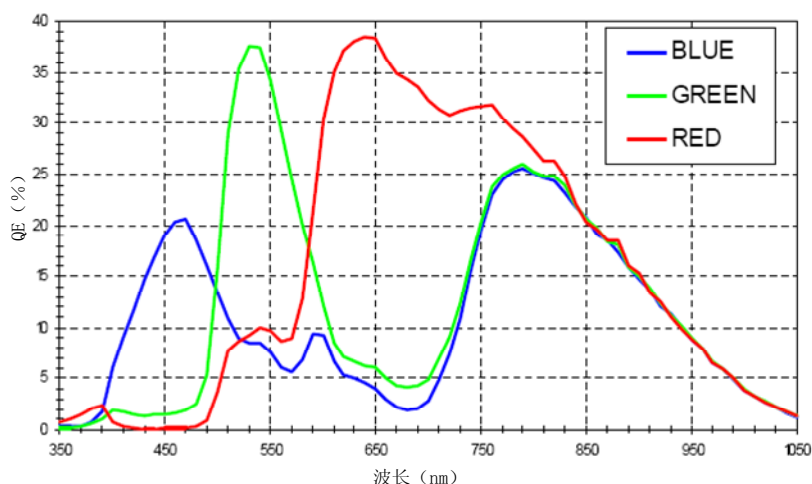


图 2.10 CMOS 图像传感器的光谱响应曲线

为了补偿这种差异，必须对图像的像素值（R、G、B）进行如公式（2.10）的变换处理。

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \bullet \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (2.10)$$

其中，系数  $b_{ij}$  是由传感器的光谱特性、光源光照条件和滤镜的光谱特性所决定，不能简单地就给出，往往需要传感器厂商在满足人眼的视觉效果的前提下，依据性能指标和测试结果综合给出。因此，这里的色彩校正不可能做到理想的效果，只能做到尽可能地减小上述的差异性。

### 2.3.3 伽马校正 (Gamma Correction)

在计算机图形领域，“伽马校正”这个术语大家并不陌生，但它的含义可能正确理解的不多。这其中又涉及到另一个术语—强度 (Intensity)，其表示的是每单位面积传播的 (光) 辐射能量。在图像显示器中，这个强度作为参量和输入的电压信号密切相关。

以目前应用最为广泛的阴极射线管显示器 CRT 为例，CRT 的感光材料的响应随着加载电子束电压信号的不同而不同。在理想状态时，输出的色彩强度 Intensity 和电子束的电压信号之间的关系应该是线性的，如图 2.11(a)所示；但实际上，如图 2.11(b)所示，输出的强度随着电压信号之间是非线性的。

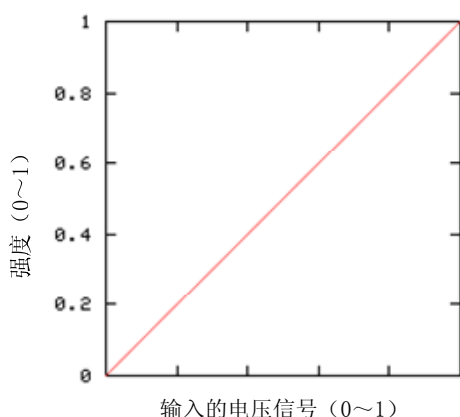


图 2.11(a) 理想的线性响应

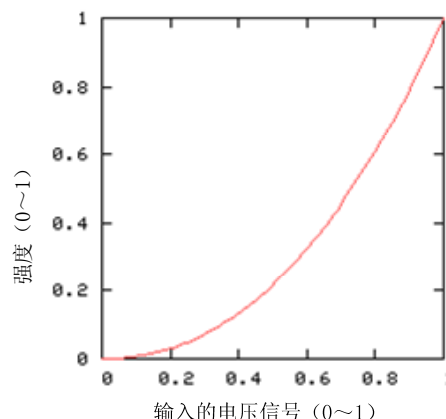


图 2.11(b) 实际的非线性响应

研究表明，显示器的输出强度和输出电压的相应大致呈幂指数关系，如公式 (2.11) 所示输出的强度随着输入电压的增长成指数增长。通常我们就把这个幂指数称为伽马 (gamma)。事实上，几乎各种显示器都存在这种非线性关系，其伽马值大小在 1.7 和 2.7 之间，CRT 的伽马值一般取 2.2。

$$I = P^\gamma \quad (2.11)$$

其中， $I$  指显示器输出的光强度； $P$  指显示器上加载的光束电压，一般光束电压  $P$  是由图像相应位置的像素值决定； $\gamma$  为伽马值。

为了在显示器上显示的图像效果和实际相符，有必要在摄像机获取图像后进行伽马校正，使得上述这种非线性校正为线性关系，公式如下：

$$P_{new} = (P_{old})^{\frac{1}{\gamma}} \quad (2.12)$$

其中， $P_{new}$  是进行了校正处理的对图像的像素值； $P_{old}$  是校正前的像素值；伽马值  $\gamma$  影响着校正的程度， $\gamma=1$  时，不进行校正， $\gamma$  越大，像素值的校正量越大。

伽马校正的具体实现方法是多种多样的，在模拟电视中，伽马校正可采用分段折线与渐变式两种。对于分段式是通过选择二极管的配置电路，确定不同的分段导通特性来实现校正处理。数字电路技术的发展，高度灵活的数字化伽马校正得到了广泛应用，可以采用数字电路的硬件实现折线式

的伽马校正电路，也可采用软件实现渐变式伽马校正电路。如图 2.12 所示，软件实现的伽马校正的步骤分为两步；1) 建立伽马校正数据表；2) 根据输入的像素值进行查表获取伽马校正后的数据。其中，伽马校正数据表的可以通过各种公式的修正公式计算获得，也可预先设定。

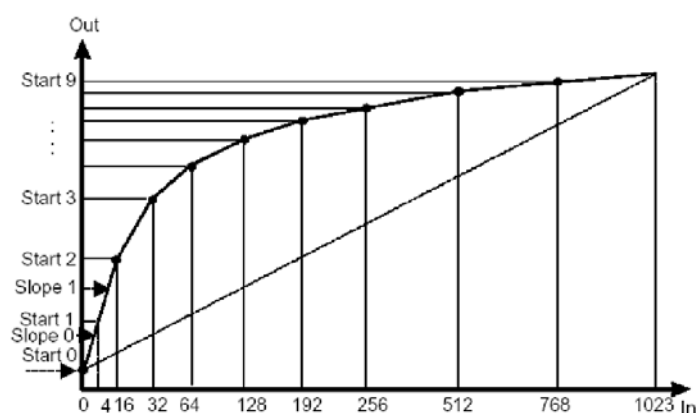


图 2.12 伽马校正示意图

### 2.3.4 图像增强 (Image Enhancement)

很显然，实际应用中图像传感器的输出图像经过上述的处理并不是完美的，图像质量获得的改进也是有限的，加之噪声、光照等原因，需要进一步处理，丢弃无用的信息，保留我们感兴趣的重要信息。图像增强作为一种重要的图像处理技术，目的无非就是两个：第一更适合人眼的感觉；第二有利于后续的分析处理。

图像增强主要包括直方图均衡、平滑滤波、中值滤波、锐化等内容。一般情况下，图像增强既可以在空间域实现，也可以在频域内实现。这里我们主要介绍在空间域内对图像进行点运算，它是一种既简单又重要的图像处理技术，它能让用户改变图像上像素点的灰度值，这样通过点运算处理将产生一幅新图像。总之，图像增强后，有利于视觉的效果和后续的处理，消除了相关性和高频噪声，有利于图像的压缩和处理，节省带宽。

#### ● 平滑滤波

图像平滑作为一种主要的图像增强技术，其主要目的是为了消除图像采样系统的质量因素所产生的噪声。噪声并不限于人眼所能看的见的失真和变形，有些噪声只有在进行图像处理时才可以发现。图像的常见噪声主要有加性噪声、乘性噪声和量化噪声等。图像中的噪声往往和信号交织在一起，尤其是乘性噪声，如果平滑不当，就会使图像本身的细节如边界轮廓、线条等变得模糊不清，如何既平滑掉噪声尽量保持图像细节，是图像平滑主要研究的任务。

一般来说，图像的能量主要集中在其低频部分，噪声所在的频段主要在高频段，同时图像中的细节信息也主要集中在其高频部分，因此，如何去掉高频干扰又同时保持细节信息是关键。为了去除噪声，有必要对图像进行平滑，可以采用低通滤波的方法去除高频干扰。图像平滑包括空域法和频域法两大类，在空域法中，图像平滑的常用方法是采用均值滤波或中值滤波，对于均值滤波，它是用一个有奇数点的滑动窗口在图像上滑动，将窗口中心点对应的图像像素点的灰度值用窗口内的各个点的灰度值的平均值代替，如果滑动窗口规定了在取均值过程中窗口各个像素点所占的权重，也就是各个像素点的系数，这时候就称为加权均值滤波；对于中值滤波，对应的像素点的灰度值用窗口内的中间值代替。在频域法中，一般采用低通滤波法。这里主要采用介绍空域处理的方法。

加权均值滤波是取一个  $n \times n$  的窗口，取该窗口内的  $n^2$  个像素的加权平均值取代中心像素原来的值。加权均值算法的一般表达形式为：

$$g(x, y) = \sum_{i=-k}^k \sum_{j=-k}^k w(i, j) f(x + i, y + j) \quad (2.13)$$



其中， $g(x,y)$ 是窗口的中心元素， $f(x+i,y+j)$ 是有噪声图像的像素， $w(i,j)$ 为加权值，其窗口大小为 $(2k+1) \times (2k+1)$ 。把  $n \times n$  的权值排成矩阵，称为加权模板。下面介绍几个  $3 \times 3$  的加权模板：

相等加权的模板为：

$$w(i,j) = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 \bullet & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (2.14)$$

锥形加权的模板为：

$$w(i,j) = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 \bullet & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad (2.15)$$

灰度差倒数加权的模板为：

$$W = \begin{bmatrix} w(x-1,y-1) & w(x-1,y) & w(x-1,y+1) \\ w(x,y-1) & w(x,y) \bullet & w(x,y+1) \\ w(x+1,y-1) & w(x+1,y) & w(x+1,y+1) \end{bmatrix} \quad (2.16)$$

其中，每个加权值为：

$$w(x+i,y+j) = \frac{d(x+i,y+j)}{2 \left[ \sum_i \sum_j d(x+i,y+j) \right]}$$

$$d(x+i,y+j) = \frac{1}{[f(x+i,y+j) - f(x,y)]}$$

图 2.13 给出了采用锥形加权的模板运算得出平滑效果图。



平滑处理前



平滑处理后

图 2.13 锥形加权模板的平滑效果图

### ● 中值滤波

中值滤波也是一种典型的低通滤波器，它的目的是保护图像的细节的同时，消除噪声。中值滤波的原理是指把以某点 $(x,y)$ 为中心的小窗口内的所有像素的灰度按从大到小的顺序排列，将中间值作为 $(x,y)$ 处的灰度值(若窗口中有偶数个像素，则取两个中间值的平均)。

对二维的数字图像，设定一个大小为 $(2k+1) \times (2k+1)$ 的窗口，计算其中值为：

$$y_{ij} = \text{median}\{x_{i+k,j+k}, i, j = -k, \dots, k\} \quad (2.17)$$

可以采用冒泡法对数组进行排序，然后返回数组元素的中值。

实际处理中可采用多种快速算法求解，例如，Narendra 提出了对图像先进行行方向的一维中值滤波，再做列方向的一维中值滤波方法，可得到去二维中值滤波相近的结果，但计算量大大降低，也易于硬件实现。T.S.Huang 提出了对图像用  $n \times n$  的滑动窗口进行中值滤波时，每次求中值只要考虑去掉最左列，补上最右列的像素，其余的像素不变，因此计算量大大缩小。

### ● 图像锐化

图像的边缘信息在图像风险和人的视觉中都是非常重要的，物体的边缘是以图像局部特性不连续的形式出现的。前面介绍的图像滤波对于消除噪声是有益的，但往往使图像中的边界、轮廓变的模糊，为了减少这类不利效果的影响，这就需要利用图像锐化技术，使图像的边缘变得更加鲜明。

图像锐化处理的目的是为了使图像的边缘、轮廓线以及图像的细节变的清晰，经过平滑的图像变得模糊的根本原因是因为图像受到了平均或积分造成的，因此可以对其进行逆运算（如微分运算）就可以使图像变的清晰。从频率域来考虑，图像模糊的实质是因为其高频分量被衰减，因此可以用高通滤波器来使图像清晰。

图像锐化的技术有两种方法：微分法和高通滤波法。这里主要介绍微分法，常用的微分锐化主要有两种：梯度锐化和拉普拉斯锐化。以拉普拉斯锐化为例，对于给定的图像  $f(x,y)$ ，其二阶差分为：

$$\begin{cases} \frac{\partial^2 f(x,y)}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y) \\ \frac{\partial^2 f(x,y)}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y) \end{cases} \quad (2.18)$$

从而拉普拉斯算子：

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} = f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \quad \text{锐化}$$

处理：

$$g(x,y) = f(x,y) - k\nabla^2 f(x,y) \quad (2.19)$$

当  $k=1$  时，等于：

$$g(x,y) = 5f(x,y) - f(x-1,y) - f(x,y+1) - f(x,y-1) \quad (2.20)$$

显然，公式（2.21）可以变成前面的模板运算，从而拉普拉斯锐化运算也变成了模板运算，其模板形式为：

$$w(i,j) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (2.21)$$

图 2.14 给出了采用锐化处理效果图。



锐化处理前



锐化处理后

图 2.14 锐化处理效果图

### ● 直方图均衡

图像直方图是图像处理中一种十分重要的图像分析工具，它描述了一幅图像的灰度级内容，从数学上来说图像直方图是图像各灰度值统计特性与图像灰度值的函数，它统计一幅图像中各个灰度级出现的次数或概率；从图形上来说，它是一个二维图，横坐标表示图像中各个像素点的灰度级，纵坐标为各个灰度级上图像各个像素点出现的次数或概率。

在介绍灰度直方图均衡之前，先讲讲直方图修正。所谓**直方图修正**，就是通过一个灰度映射函数  $S=F(r)$ ，将原灰度直方图改造成你所希望的直方图。所以，直方图修正的关键就是灰度映射函数。**直方图均衡化**是一种最常用的直方图修正。它是把给定图像的直方图分布改造成均匀直方图分布。由信息学的理论来解释，具有最大熵(信息量)的图像为均衡化图像。

假定图像的总像素数目为  $n$ ，而某个灰度级  $k$  的像素数目为  $m_k$ ，该灰度级的概率密度为：

$$P_r(r_k) = \frac{n_k}{n} \quad (2.22)$$

则图像直方图均衡的变化函数为：

$$S_k = T(r_k) = \sum_{j=0}^k \frac{n_j}{n} = \sum_{j=0}^k P_r(r_j) \quad (2.23)$$

$$k = 0, 1, 2, \dots, L-1$$

图 2.15 给出了采用直方图均衡的效果图。



直方图均衡前

直方图均衡后

图 2.15 直方图均衡的效果图

### 2.3.5 白平衡 (White Balance)

白平衡作为图像处理的一个重要术语，也随着数码相机的普及进入了人们的认识中。白平衡指



的就是对白色物体的还原。当我们用肉眼观看这大千世界时，在不同的光线下，对相同的颜色的感觉基本是相同的，比如在早晨旭日初升时，我们看一个白色的物体，感到它是白的；而我们在夜晚昏暗的灯光下，看到的白色物体，感到它仍然是白的。这是由于人类从出生以后的成长过程中，人的大脑已经对不同光线下的物体的彩色还原有了适应性。但是，图像传感器没有这种人眼的适应性，在不同的光线下，由于图像传感器输出的不平衡性，造成其输出的彩色失真：或者图像偏蓝，或者偏红，如图 2.16 所示。



正常图像



色温高



色温低

图 2.16 白平衡示意图

理解白平衡，涉及到另一个重要的概念：色温。所谓色温，简而言之，就是定量地以开尔文温度表示色彩。**色温越高，蓝色成分就越多；色温越低，红色成分就越多**，在摄影、摄像时，不同色温光源下拍摄物体，获得的图像不可避免会出现色彩上的偏差。为了很获得现实世界中各种色彩的图像，必须消除环境中光源色温的影响，即进行白平衡处理。

传统的白平衡方法，首先在色温环境中拍摄一纯白色物体，分析所拍摄的图像数据，对白色物体的数据进行平均。得出三原色的平均值（ $R_{\text{mean}}$ 、 $G_{\text{mean}}$ 、 $B_{\text{mean}}$ ），根据白色的定义：

$$R=G=B$$

改变 R、B 感应通道的增益可以实现图像的白平衡，这种白平衡方法需要有白色参照物，使用不便。因此，实际应用中，产生了一些**自动白平衡的算法**，主要有：

#### 1) 全局平衡法

认为所拍摄的图像的 RGB 三色分量的统计平均应该相等，对于拍摄的图像进行统计平均，以 R、B 分量的均值作为白平衡校准的依据。

#### 2) 局部白平衡法

搜索所拍摄的图像中，最亮的区域作为白色区域，该区域的 RGB 三色分量的统计平均值应该相同，以该区域的 R、B 分量的均值作为白平衡校准的依据。

2.4 视频质量

对压缩后的视频质量估计是一件困难的工作。大体上，可分为主观视频质量评定和客观视频质量评定两种估计方法。

2.4.1 主观质量的评定

由于个人的视觉系统（HVS）不尽相同，对视频内容的熟悉程度也不一样。为了减少主观随意性，在对视频图像主观评定前，选若干名专家和“非专家”作为评分委员，共同利用五项或七项评分法对同一种视频图像进行压缩编码构图像评定。最后按加权平均法则对该压缩后的图像质量进行主观评定，如表 2.2 所示。

表 2.2 主观评价分数标准

CCIR 五级评分等级	评分等级	高清晰度采用七级评分等级	评价
	7	不能觉察任何图像损伤	特别好
	6	刚能觉察有图像损伤	相当好
优	5	不同程度的觉察，轻度损伤	很好
好	4	有损伤，但不令人讨厌	好
稍差	3	有令人讨厌损伤	稍差
很差	2	损伤令人讨厌，但尚可忍受	很差
劣	1	非常令人讨厌损伤，无法观看	劣

测试方法可用随机次序请评委观察比较原始图像和压缩编码的图像。国际上称为 DSCOS 的测试系统如图 2.17 所示。其中 A 为原始图像，B 为编码解码后的图像，以任意的 A、B 次序让评委打分评定。

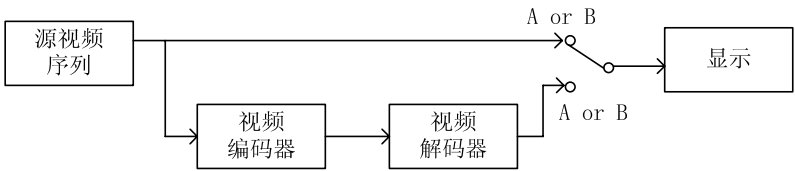


图 2.17 DSCQS 测试系统

2.4.2 客观质量的测量

主观的视频质量评分更接近人的真实视觉感受，但需耗费人力和时间，成本较高。客观质量的测定方法速度快、易实行，但往往不会太符合人眼的视觉感受，只能说大体上的质量。客观质量测定方法应致力于改进其测试标准和测试方法，使其符合人的视觉感受。

最常用的测试标准是峰值信号与噪声之比（PSNR）：

$$PSNR_{dB}=10 \log_{10}(2^n-1)^2 / MSE$$
 (2.24)

其中 MSE 为原始和编解码后图像之间的均方误差， $(2^n-1)^2$  为图像种最大可能的信号值平方，n 为表示每个像素的比特数。

一般讲，PSNR 愈高视频质量愈高；反之亦然。但实际上有时并非如此，如图 2.18 和图 2.19 所示。图 2.19 的 PSNR=27.7dB，其主观评定可能比图 2.18（b）（c）的高，但客观质量 PSNR 却低于图 2.18（b）（c）的 30.6dB 和 28.3dB。这是因为图 2.19 中的脸部更清晰，只是背景模糊，而人眼对脸部往往更敏感更重视。



图 2.18 PSNR 举例 (a) 原始; (b) 30.6dB; (c) 28.3dB



图 2.19 背景模糊图像 27.7dB

## 参考文献

- 1 Iain E.G.Richardson, H.264 and MPEG-4 Video Compression Video Coding for Next Generation Multimedia, Wiley Press , 2003
- 2 孙景鳌, 蔡安妮, 彩色电视基础, 人民邮电出版社, 1996
- 3 许志祥, 数字电视与图像通信, 上海大学出版社, 2000
- 4 张兆扬, 陈加卿, 徐在方, 数字电视原理, 科学出版社, 1987.12
- 5 <http://www-ise.stanford.edu/~tingchen/main.htm>
- 6 Color Correction for Image Sensors, Kodak Image Sensor Solution, 2003.10
- 7 <http://graphics.stanford.edu/gamma.html>
- 8 <http://www.teamten.com/lawrence/graphics/gamma/>
- 9 钟志光, 卢君, 刘伟荣, Visual C++.NET 数字图像处理实例与解析, 清华大学出版社, 2003.5
- 10 胡波, 林青, 陈光梦, 基于先验知识的自动白平衡, 电路与系统学报, 2001.6

## 第3章 视频压缩编码的基本原理

### 3.1 预测编码

#### 3.1.1 预测编码的基本概念

预测法是最简单和实用的视频压缩编码方法，这时压缩编码后传输的并不是像素本身的取样幅值，而是该取样的预测值和实际值之差。

为什么取像素预测值与实际值之差作为传输的信号？因为大量统计表明，同一幅图像的邻近像素之间有着相关性，或者说这些像素值相似。邻近像素之间发生突变或“很不相似”概率很小。而且同帧图像中邻近行之间对应位置的像素之间也有较强的相关性。人们可以利用这些性质进行视频压缩编码。

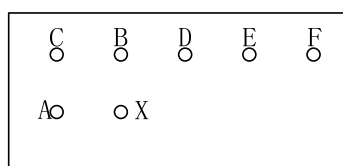


图 3.1 邻近像素间的相关性

例如，同一帧内邻近像素，当前像素为 X，其左邻近像素为 A，上邻近像素为 B，上左邻近像素为 C 等。显然与 X 之间的距离近的像素，如 A 和 B 与 X 的相关性强，愈远相关性愈弱，如 C、D、E、F 等像素。以 P 作为预测值，按与 X 的距离不同给以不同的权值，把这些像素的加权和作为 X 的预测值，与实际值相减，得到差值 q。由于临近像素之间相关性强，q 值非常小，达到压缩编码的目的。

接收端把差值 q 与预测值（事先已定义好，比当前 X 早到达接收端像素，如 A）相加，恢复原始值 X。归纳如下：

编码端： $X - A = q$

解码端： $q + A = X$

按以上原理可得预测编码框图，如图 3.2 所示。这种预测编码也称为差分脉冲编码（DPCM）。

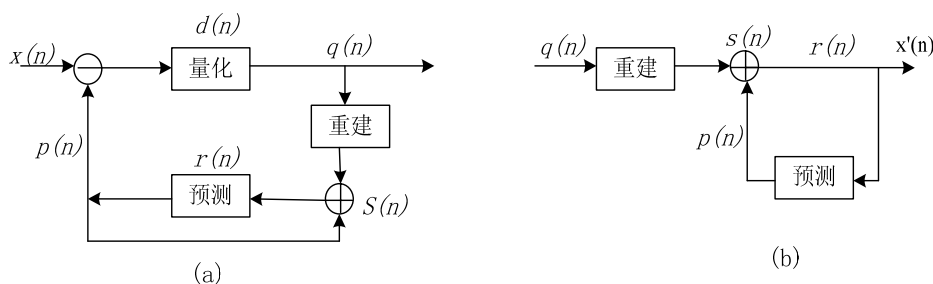


图 3.2 预测编码

其中， $x(n)$  为当前像素的实际值， $p(n)$  为其预测值， $d(n)$  为差值或残差值。该差值经量化后得到残差量化值  $q(n)$ 。预测值  $p(n)$  经预测器得到，预测器输入为已存储在预测器内前面的各像素，和当前值，它们的加权和即为下一个预测器输出。

由图 3.2 可见，**解码输出  $x'(n)$  与原始信号  $x(n)$  之间有个因量化而产生的量化误差。**

现在进一步说明预测法可压缩视频信息的理由。大量统计表明，由于相关性的存在，邻近像素值之差很小。其差值信号的概率分布如图 3.3 所示。可见该差值信号的方差是比较小的。由于图像的误差信号  $d(n)$  方差相对图像信号本身方差较小，其量化器的动态范围可以缩小，相应的量化分层数目就可减少，每个像素的编码比特数也显著下降，而且不致视频质量明显降低，达到视频压缩的



目的。

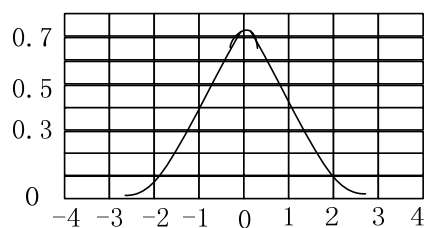


图 3.3 图像差值信号的概率分布

### 3.1.2 帧内预测编码

#### 3.1.2.1 一维最佳预测

现在按上述的概念重新画成如图 3.4 所示的预测编码器。

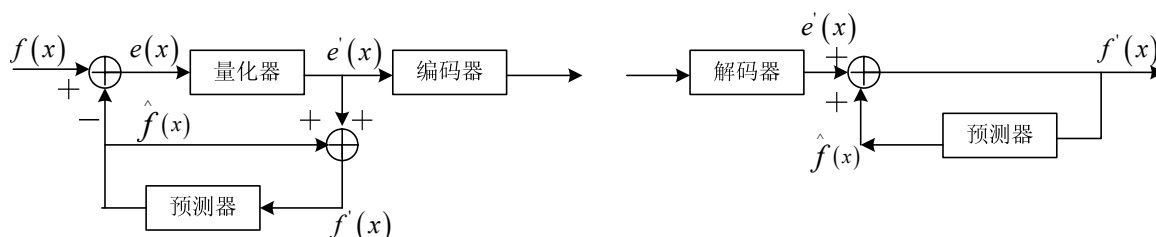


图 3.4 一维预测编码器

预测模型可以是一维的，也可以是二维或多维的；可以是线性的，也可非线性。下面先讨论一维线性预测方法。

设预测值为

$$\hat{f}(x) = \sum_{k=1}^m a_k f(x-k) \quad (3.1)$$

其中  $f(x)$  为当前像素值，以  $f(x)$  为基准， $f(x-1)$  为前一个像素值， $f(x-2)$  为前两个像素值。把前  $m$  个像素值的加权和  $\hat{f}(x)$  作为  $f(x)$  的预测值。 $a_k$  是预测系数， $m$  为预测阶数。其预测误差如下：

$$e(x) = f(x) - \hat{f}(x) = f(x) - \sum_{k=1}^m a_k f(x-k) \quad (3.2)$$

$e(x)$  经量化得到  $e'(x)$ ，为获得最佳预测效果，应使  $e(x)$  的均方误差最小，编码效率最高：

$$\sigma_e^2 = E\{e^2(x)\}$$

$$\text{令 } \frac{\partial \sigma_e^2}{\partial a_j} = 0 \quad (3.3)$$

设  $E\{f(x-j)f(x-k)\} = R_{k-j}$  为信号间的相关函数，因信号是平稳信源，即像素间自相关是常数，与位置无关，像素间相关性对称，可得出：

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix} = \begin{bmatrix} R_0 & R_1 & \dots & R_{m-1} \\ R_1 & R_0 & \dots & R_{m-1} \\ \vdots & \vdots & \ddots & \vdots \\ R_{m-1} & \dots & R_1 & R_0 \end{bmatrix}^{-1} \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_m \end{bmatrix} \quad (3.4)$$

即预测系数：
$$\alpha = [R]^{-1} * r \quad (3.5)$$

可以证明，这时预测误差的方差为：

$$\sigma_e^2 = R_0 - \sum_{k=1}^m \alpha_k R_k = \sigma_x^2 - \sum_{k=1}^m \alpha_k R_k \quad (3.6)$$

可见，相关性  $R_k$  越大，预测误差方差  $\sigma_e^2$  越小于信号方差，压缩效率也就越高。

### 3.1.2.2 二维最佳预测

二维预测编码器框图如图 3.5 所示。

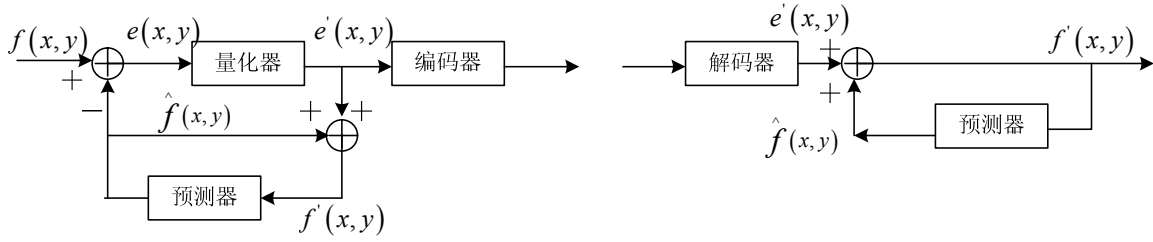


图 3.5 二维预测编码

图 3.5 中  $f(x, y)$  为当前像素值， $(x, y)$  为其所在水平和垂直位置的  $x, y$  坐标值。由一维预测可推广，其二维预测值为：

$$\hat{f}(x, y) = \sum_{(k, l) \in Z} \alpha_{k, l} f(x - k, y - l) \quad (3.7)$$

$\alpha_{k, l}$  为二维预测系数， $Z$  为预测区域， $(k, l)$  分别为对当前点进行预测点像素的水平和垂直位置坐标值。设  $f(x, y)$  为二维平稳随机过程，预测误差为：

$$e(x, y) = f(x, y) - \hat{f}(x, y) \quad (3.8)$$

$$\text{令 } \frac{\partial \sigma_e^2}{\partial a_j} = 0$$

得：

$$R(i, j) - \sum_{(k, l) \in Z} \alpha_{k, l} R(x - i, y - j) = 0 \quad (3.9)$$

由二维图像得相关函数，可得其预测误差方差：

$$\sigma_e^2 = R(0, 0) - \sum_{(k, l) \in Z} \alpha_{k, l} R(k, l) \quad (3.10)$$

为了利用数字电路实现预测系数，往往将其取为 1/2、1/4、1/8 之类的分数表示。有人提出了一下预测值：

$$\hat{f}(x, y) = \frac{1}{2} f(x, y - 1) + \frac{1}{8} f(x - 1, y - 1) + \frac{1}{4} f(x - 1, y) + \frac{1}{8} f(x - 1, y + 1) \quad (3.11)$$

其中各预测点得位置如图 3.6 所示。

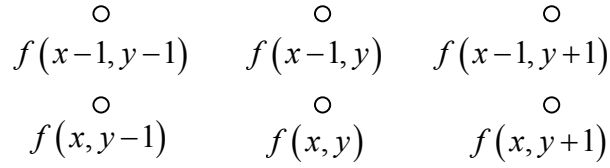


图 3.6 二维图像像素位置

### 3.1.2.3 预测编码增益

视频预测编码的效果是得到了视频的压缩编码，即相对于不压缩的 PCM 而言，每个像素值被分配的比特数被减少。换言之，可用相同比特率下，产生量化失真之比来定义编码增益。

由文献 3 可知，对于 PCM，其率失真  $D_{PCM}$  为：

$$D_{PCM} = \varepsilon^2 \sigma_f^2 2^{-2R} \quad (3.12)$$

其中  $\sigma_f^2$  为原始信号的方差， $\sigma_{PCM}^2$  为该信号量化误差的方差， $\varepsilon$  为量化误差的概率密度函数。如预测误差采用最佳标量量化，量化失真  $D_{DPCM}$  与码率 R 之间关系为：

$$D_{DPCM} = \varepsilon_p^2 \sigma_\rho^2 2^{-2R} \quad (3.13)$$

其中， $\sigma_\rho^2$  为预测误差的方差， $\varepsilon_p^2$  是预测误差的概率密度函数，预测编码增益为：

$$G_{DPCM} = \frac{D_{PCM}}{D_{DPCM}} = \frac{\varepsilon^2 \sigma_f^2}{\varepsilon_p^2 \sigma_\rho^2} \quad (3.14)$$

对于高斯信源， $\varepsilon^2$  和  $\varepsilon_p^2$  两者可以认为相等，于是得：

$$G_{DPCM} = \frac{\sigma_f^2}{\sigma_\rho^2} \quad (3.15)$$

例：设一幅图像中每个  $2 \times 2$  块，进行预测编码（图 3.7），其中  $\rho_h$ 、 $\rho_v$ 、 $\rho_d$  分别为相邻像素之间的

相关系数，设  $\rho_h = \rho_v = \rho = 0.95$ ， $\rho_d = \rho^2 = 0.9025$ ，求  $G_{DPCM}$ 。

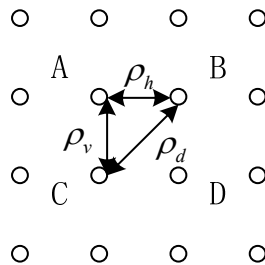


图 3.7  $2 \times 2$  块

解：由前面推导可知，

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \rho \\ \rho \\ \rho^2 \end{bmatrix}$$

$$D_{DPCM} = \varepsilon_p^2 \sigma_\rho^2 2^{-2R}$$

$$G_{DPCM} = \frac{\sigma_f^2}{\sigma_\rho^2} = \frac{1}{(1-\rho^2)^2} = 105.19 \quad (3.16)$$

#### 3.1.2.4 预测编码的量化器

图 3.3 给出了大量统计后差值信号的概率分布，一般说图像中平坦区域比突变区域多得多，例如人脸中，只有眼睛、鼻子、嘴等少量地方细节出现，其余则为平坦或缓变区域。

人眼视觉特性实验表明，在亮度突变部分或变化大的部分，量化误差大些不会使人眼敏感，可采取粗量化，量化节距可取大一些，这时虽然需多些比特数，但面积小总比特数不大；反之，在亮度变化缓慢区域，则应取细量化，但由于平坦区域  $e(x,y)$  小，也不会增加很多比特数。总之，利用人眼这种掩盖效应采用非线性（不均匀）量化，可使总码率有所下降。表 3.1 为量化器输入输出值举例。由于误差信号值有正有负，总量化级数为 19。

表 3.1 量化值举例

输出量化值	0	3	6	11	18	29	46	71	110	150
输入量化值	0~1	2~4	5~8	9~14	15~23	24~37	38~58	59~90	91~139	140~225

由于量化，预测编码会产生过载、颗粒噪声、伪轮廓以及边沿忙乱等，如图 3.8 所示。它们都是由于量化值过小或不够小及像素变化过快等跟不上变化造成的。

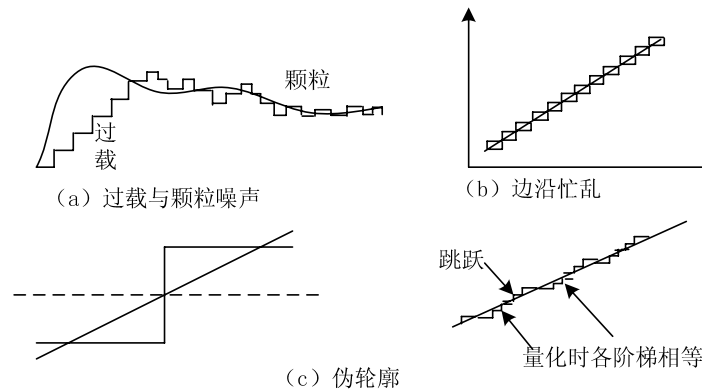


图 3.8 预测编码的各种噪声

#### 3.1.2.5 二维预测编码器

图 3.9、3.10 为一具体的二维预测编码器举例，设原数字序列为  $f(x,y)$ ，预测公式为：

$$\hat{f}(x,y) = \frac{15}{32}f(x,y-1) + \frac{1}{8}f(x-1,y-1) + \frac{9}{32}f(x-1,y) + \frac{1}{8}f(x-1,y+1) \quad (3.17)$$

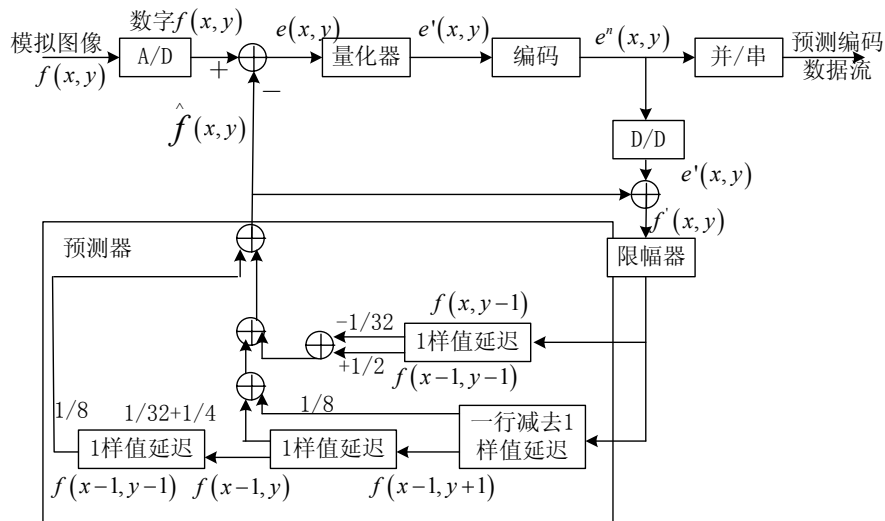


图 3.9 二维预测编码器

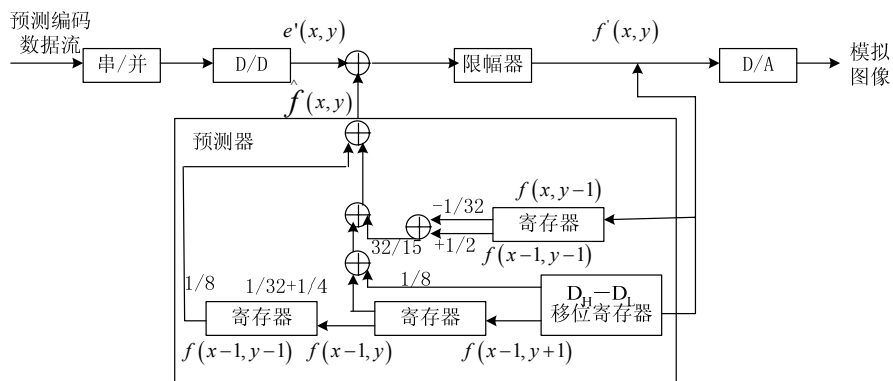


图 3.10 二维预测解码器

### 3.1.3 帧间预测编码

一般而言，帧间预测编码编码效率比帧内更高。有人测得，对缓慢变化 256 级灰度的黑白图像序列，帧间差超过阈值 3 的像素不到一帧像素的 4%；对剧烈变化 256 亮度值的彩色电视序列，帧间差超过阈值 6 的像素平均只占一帧的 7.5%。下文将讨论单向预测、双向预测、重叠块运动补偿三个内容。

#### 3.1.3.1 单向预测

##### (1) 预测原理

图 3.11 为单向预测的预测编码框图。

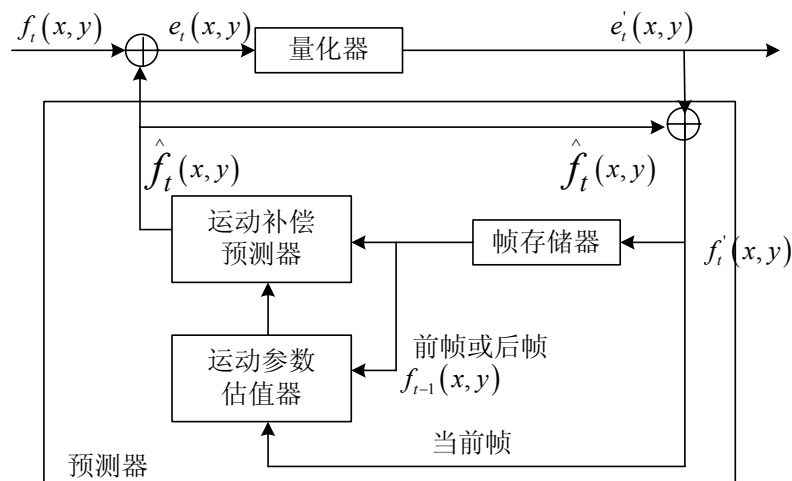


图 3.11 单向预测帧间编码

当前帧图像  $f_t(x, y)$  与预测图像  $\hat{f}_t(x, y)$  相减后的帧误差  $e_t(x, y)$ ，经量化器量化后输出  $e_t'(x, y)$ ，传送到信道。预测图像  $\hat{f}_t(x, y)$  与  $e_t'(x, y)$  相加，得  $f_t'(x, y)$ ，当不计量化失真时， $f_t'(x, y)$  即当前的  $f_t(x, y)$ 。

把当前帧  $f_t'(x, y)$  与帧存储器输出的前一帧  $f_{t-1}(x, y)$  (也称参考帧) 同时输入运动参数估值器，经搜索、比较得到运动矢量  $MV$ 。此  $MV$  输入运动补偿预测器，得到预测图像  $\hat{f}_t(x, y)$ 。预测图像  $\hat{f}_t(x, y)$  不可能完全等同于当前图像  $f_t(x, y)$ ，无论预测得如何精确，总存在帧误差  $e_t(x, y)$ 。

由上述可知，利用上一帧的图像经运动矢量位移作为预测值的方法称为单向预测或单向时间预测。这时，

$$\hat{f}_t(x, y) = f_t(x + i, y + j) \quad (3.18)$$

其中， $(i, j)$  即运动矢量。

如何减小帧差和更精确预测当前像素是提高帧间压缩编码效率的关键之处。

## (2) 基于块匹配算法的运动矢量估计

上述原理以像素为单位进行预测，除了传送帧差外，还增加了每个像素的运动矢量，编码效率显著下降。

实际上，两帧之差的物体运动一般是刚体的平移运动，位移量不大，因此往往把一帧图像分成若干  $M \times N$  块，以块为单位分配运动矢量，大大降低总码率。如图 3.12 所示。

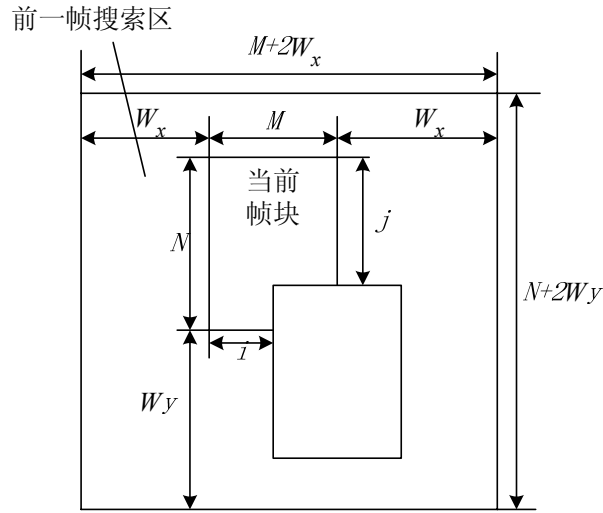


图 3.12 块匹配算法

设前一帧搜索区为  $(M+2W_x, N+2W_y)$ ，当前帧块与前一帧块的位移为  $d(i, j)$ ，在搜索区中，如能找到与当前帧块匹配的前一帧块，则该  $d(i, j)$  即为所需的运动矢量。

常用的匹配准则有：

a) 均方误差 (MSE) 最小准则：

$$MSE(i, j) = \frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N [f_t(x, y) - f_{t-1}(x+i, y+j)]^2 \quad (3.19)$$

b) 绝对误差均值 (MAD) 最小准则：

$$MAD(i, j) = \frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N |f_t(x, y) - f_{t-1}(x+i, y+j)| \quad (3.20)$$

### (3) 搜索方法

#### a 穷尽搜索法

穷尽搜索法对搜索窗内的每一点都用匹配准则进行计算，找到 MSE 或 MAD 最小时的点  $(i, j)$  值，作为所需的运动矢量  $d(i, j)$ 。

该法计算量大，如采用 MAD 准则，需计算  $(2W_x+1) \times (2W_y+1)$  个 MAD 值，但它能够找到全局最优运动矢量。

#### b 快速搜索法

为了更快找到  $d(i, j)$ ，可采用一些快速搜索法，这里只以三步搜索法为例加以说明。如图 3.13 所示。

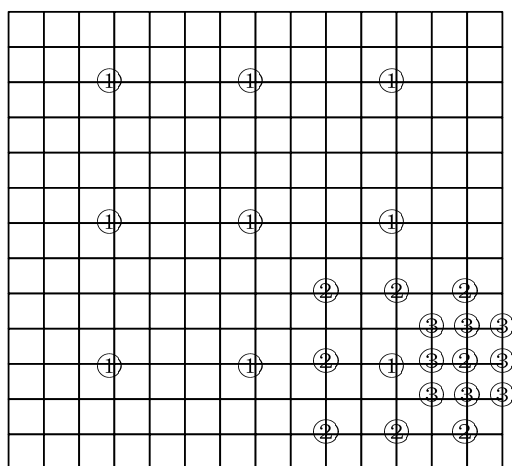


图 3.13 三步搜索法

一个  $16 \times 16$  搜索区中心点  $(i,j)=(0,0)$ ，以搜索区最大搜索长度的一半为步长，计算中心点及周围 8 个邻近点的 MAD 值，如找到某个点 MAD 最小，再以该点为中心，步长减为原来的一半，依次类推，到了第三步，再把步长减半，计算 MAD 值，其中 MAD 最小点运动矢量即为所求的值。快速算法的速度快，但不能保证全局最优。

### 3.1.3.2 双向预测

有时，不只是利用前一帧像素预测，还需利用后一帧像素，即预测值为：

$$\hat{f}_t(x,y) = \alpha_{t-1} f_t(x+i,y+j) + \alpha_{t+1} f_t(x+i',y+j') \quad (3.21)$$

其中  $d(i,j)$  和  $d(i',j')$  分别为  $t$  到  $t-1$  和  $t$  到  $t+1$  间的运动矢量 MV， $\alpha_{t-1}$  和  $\alpha_{t+1}$  分别为前向和后向预测系数，可按最佳预测公式确定。

这时，前向参考帧预测当前帧称为前向运动补偿，利用后向参考帧预测当前帧称为后向运动补偿，利用前后向同时预测的就称为双向预测运动补偿。

双向预测在实时通信中是不能应用的，例如会议电视、可视电话等，因为后向预测在当前帧之后进行，会引入编码时延。它可用在广播电视系统中，如采用 MPEG 标准的编码系统，**特别针对一些暴露区域，即  $t-1$  帧尚未暴露而  $t+1$  帧已呈现出来的区域**。图 3.14 为单向和双向预测编码举例。

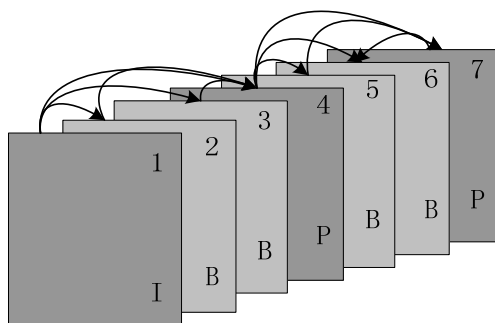


图 3.14 单向和双向预测举例

为了进一步提高编码效率，多帧预测（包括单向和双向预测）被引入，**如 H.264 标准参考帧可达 5~15 帧**。

### 3.1.3.3 重叠块运动补偿 OBMC

以上基于块的运动补偿从计算量上看是比较简单，但这种人为的块划分使得每个块由一个运动



矢量，容易产生方块效应，特别是运动矢量估计不准确或物体运动非简单的平移运动及一个块中有几个不同运动物体时。OBMC 方法解决了运动矢量估计不准确的方法。

采用 OBMC 时，一个像素的预测不仅基于它所属的 MV 估计，还基于其相邻的 MV 估计。如图 3.15 所示，以 4 个邻块为例。

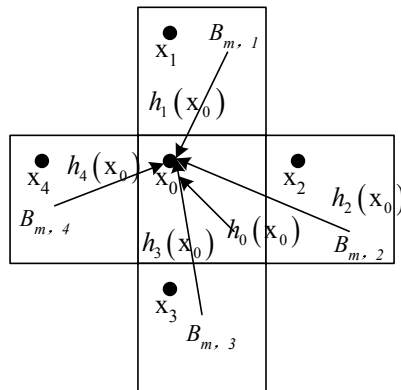


图 3.15 4 邻块的 OBMC

这时预测值为：

$$\hat{f}_p(x, y) = \sum_{k=1}^4 h_k(x_k, y_k) f_r(x_k + i_k, y_k + j_k) \quad (3.22)$$

为找到最佳预测估计，应使下式最小：

$$E \left\{ \left| f(x, y) - \sum_{k=1}^4 h_k f_r(x_k + i_k, y_k + j_k) \right|^2 \right\} \quad (3.23)$$

为保持图像的均值，还需加上如下约束：

$$\sum_{k=1}^4 h_k(x_k, y_k) = 1 \quad (3.24)$$

可用迭代法进行运动估计和 OBMC。

### 3.1.4 运动估计

#### 3.1.4.1 基本概念

在帧间预测编码中，由于活动图像邻近帧中的景物存在着一定的相关性。因此，可将活动图像分成若干块或宏块，并设法搜索出每个块或宏块在邻近帧图像中的位置，并得出两者之间的空间位置的相对偏移量，得到的相对偏移量就是通常所指的运动矢量，得到运动矢量的过程被称为运动估计。

运动矢量和经过运动匹配后得到的预测误差共同发送到解码端，在解码端按照运动矢量指明的位置，从已经解码的邻近参考帧图像中找到相应的块或宏块，和预测误差相加后就得到了块或宏块在当前帧中的位置。

通过运动估计可以去除帧间冗余度，使得视频传输的比特数大为减少，因此，运动估计是视频压缩处理系统的一个重要组成部分。本节先从运动估计的一般方法入手，重点讨论了运动估计的三个关键问题：将运动场参数化、最优化匹配函数定义以及如何寻找到最优化匹配。

#### 3.1.4.2 运动估计的方法

一般的运动估计方法如下：设  $t$  时刻的帧图像为当前帧  $f(x, y)$ ， $t'$  时刻的帧图像为参考帧  $f'(x,$

y)，参考帧在时间上可以超前或者滞后于当前帧，如图 6.12 所示，当  $t' < t$  时，称之为后向运动估计，当  $t' > t$  时，称之为前向运动估计。当在参考帧  $t'$  中搜索到当前帧  $t$  中的块的最佳匹配时，可以得到相应的运动场  $d(x; t, t \pm \Delta t)$ ，即可得到当前帧的运动矢量。

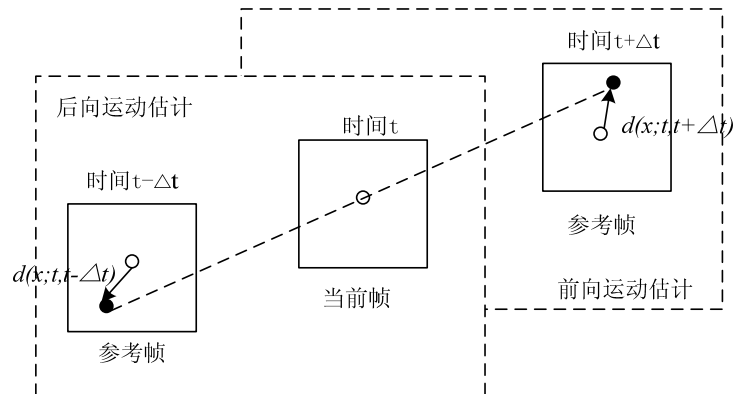


图 6.12 前向和后向运动估计

H.264 编码标准和以往采用的视频压缩标准很大的不同在于，在运动估计过程中采用了多参考帧预测来提高预测精度，多参考帧预测就是在编解码端建一个存储  $M$  个重建帧的缓存，当前的待编码块可以在缓存内的所有重建帧中寻找最优的匹配块进行运动补偿，以便更好地去除时间域的冗余度。由于视频序列的连续性，当前块在不同的参考帧中的运动矢量也是有一定的相关性的。假定当前块所在帧的时间为  $t$ ，则对应前面的多个参考帧的时间分别为： $t-1, t-2, \dots$ 。则当在帧  $t-2$  中搜索当前块的最优匹配块时，可以利用当前块在帧  $t-1$  中的运动矢量  $MV_{NR}$  来估测出当前块在帧  $t-2$  的运动矢量。

### 3.1.4.3 运动表示法

由于在成象的场景中一般有多个物体作不同的运动，如果直接按照不同类型的运动将图像分割成复杂的区域是比较困难的。最直接和不受约束的方法是在每个像素都指定运动矢量，这就是所谓**基于像素表示法**。这种表示法是对任何类型图像都是适用的，但是它需要估计大量的未知量，并且它的解时常在物理上是不正确，除非在估计过程中施加适当的物理约束。这在具体实现时是不可能的，通常采用基于块的物体运动表示法。

#### 3.1.4.3.1 基于块的运动表示法

一般对于包含多个运动物体的景物，实际中普遍采用的方法是把一个图像帧分成多个块，使得在每个区域中的运动可以很好地用一个参数化模型表征，这被称为块匹配法，即将图像分成若干个  $n \times n$  块（典型值： $16 \times 16$  宏块），为每一个块寻找一个运动矢量  $MV$ ，并进行运动补偿预测编码。

每一个帧间宏块或块都是根据先前已编码的数据预测出的，根据已编码的宏块、块预测的值和当前宏块、块作差值，结果被压缩传送给解码器，与解码器所需要的其他信息如（运动矢量、预测模型等）一起用来重复预测过程。

每个分割区域都有其对应的运动矢量，并必须对运动矢量以及块的选择方式进行编码和传输。在细节比较多的帧中如果选择较大的块尺寸，意味着用于表明运动矢量和分割区域类型的比特数会少些，但是运动压缩的冗余度要多一些；如果选择小一点的块尺寸，那么运动压缩后冗余度要少一些，但是所需比特数要多一些。因此必须要权衡块尺寸选择上对压缩效果的影响，一般对于细节比较少、比较平坦的区域选择块尺寸大一些，对于图像中细节比较多的区域选择块尺寸小一些。

宏块中的每个色度块 (Cb 和 Cr) 尺寸宽高都是亮度块的一半，色度块的分割方法和亮度块同样，只是尺寸上宽高都是亮度块一半（如亮度块是  $8 \times 16$  块尺寸大小，那么色度块就是  $4 \times 8$ ，如果亮度块

尺寸为 $8 \times 4$ ，那么色度块便是 $4 \times 2$ 等等)。每个色度块的运动矢量的水平和垂直坐标都是亮度块的一半。

#### 3.1.4.3.2 亚像素位置的内插

帧间编码宏块中的每个块或亚宏块分割区域都是根据参考帧中的同尺寸的区域预测得到的，它们之间的关系用运动矢量来表示。

由于自然物体运动的连续性，相邻两帧之间的块的运动矢量不是以整像素为基本单位的，可能真正的运动位移量是以 $1/4$ 像素或者甚至 $1/8$ 像素等亚像素作为为单位的。

图 3.17 给出了一个视频序列当采用 $1/2$ 像素精度、 $1/4$ 像素精度和 $1/8$ 像素精度时编码效率的情况，从图中可以看到 $1/4$ 像素精度相对于 $1/2$ 像素精度的编码效率有很明显的提高，但是 $1/8$ 像素精度相对于 $1/4$ 像素精度的编码效率除了在高码率情况下并没有明显的提高，而且 $1/8$ 像素的内插公式更为复杂，因此在 H.264 的制定过程中 $1/8$ 像素精度的运动矢量模型逐渐被取消而只采用了 $1/4$ 像素精度的运动矢量模型。

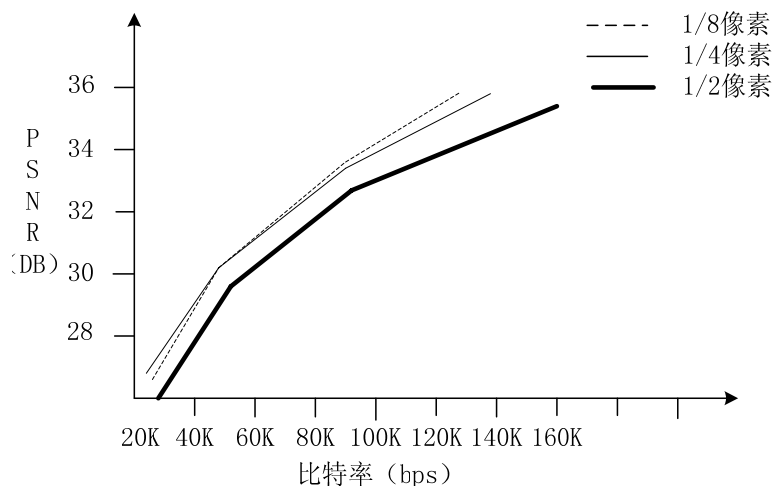


图 3.17 高精度的亚像素运动搜索对编码效率的影响

#### 3.1.4.3.3 运动矢量在时空域的预测方式

如果对每个块的运动矢量进行编码，那么将花费相当数目的比特数，特别是如果选择小尺寸的块的时候。由于一个运动物体会覆盖多个分块，所以空间域相邻块的运动矢量具有很强的相关性，因此，每个运动矢量可以根据邻近先前已编码的块进行预测，预测得到的运动矢量用  $MV_p$  表示，当前矢量和预测矢量之间的差值用  $MVD$  表示。同时由于物体运动的连续性，运动矢量在时间域也存在一定相关性，因此也可以用邻近参考帧的运动矢量来预测。

##### 1) 运动矢量空间域预测方式

###### a、运动矢量中值预测 (Median Prediction)

利用与当前块 E 相邻的左边块 A，上边块 B 和右上方的块 C 的运动矢量，取其中值来作为当前块的预测运动矢量。

设 E 为当前宏块、宏块分割或者亚宏块分割，A 在 E 的左侧，B 在 E 的上方、C 在 E 的右上方，如果 E 的左侧多于一个块，那么选择最上方的块作为 A，在 E 的上方选择最左侧的块作为 B。图 3.18 表示所有的块尺寸相同，图 3.19 表示邻近块尺寸不同时作为预测 E 的运动矢量的块的选择。

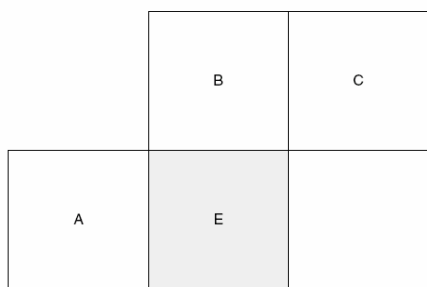


图 3.18 块尺寸相同的当前块和邻近块

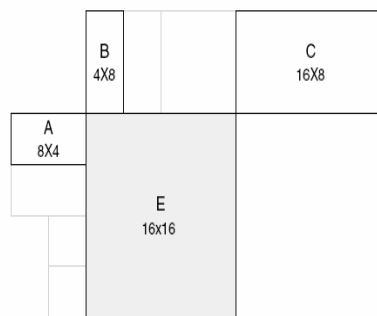


图 3.19 块尺寸不同的当前块和邻近块

在预测 E 的过程中遵守以下准则：

- 1、对于除了块尺寸为  $16 \times 8$  和  $8 \times 16$  的块来说， $MV_p$  是块 A、B 和 C 的运动矢量的中值；
- 2、对于  $16 \times 8$  块来说，上方的  $16 \times 8$  块的  $MV_p$  根据 B 预测得到，下方的  $16 \times 8$  块的  $MV_p$  根据 A 得到；
- 3、对于  $8 \times 16$  块来说，左侧的  $16 \times 8$  块的  $MV_p$  根据 A 预测得到，右侧的  $16 \times 8$  块的  $MV_p$  根据 C 得到；
- 4、对于不用编码的可跳过去的宏块， $16 \times 16$  矢量  $MV_p$  如第一种情况得到。

#### b、空间域的上层块模式运动矢量（Uplayer Prediction）

H.264 标准中提供的块尺寸有  $16 \times 16$ ， $8 \times 16$ ， $16 \times 8$ ， $8 \times 8$ ， $8 \times 4$ ， $4 \times 8$ ， $4 \times 4$ ，它们的图象分割区域分别定义为搜索模式  $Mode1$ - $Mode7$ 。假设当前分割区域的预测模式为  $Mode_{curr}$ ，上层模式  $Mode_{up}$  定义为：

$$Mode_{Up} = \begin{cases} unavailable, & \text{if } Mode_{curr} == Mode1 \\ Mode1, & \text{if } Mode_{curr} == Mode2, Mode3 \\ Mode2, & \text{if } Mode_{curr} == Mode4 \\ Mode4, & \text{if } Mode_{curr} == Mode5, Mode6 \\ Mode5, & \text{if } Mode_{curr} == Mode7 \end{cases} \quad (3.25)$$

如图 3.20 所示为利用空间域上层搜索模式的运动矢量作为当前块的预测运动矢量的预测方式。

$$MV_{pred\_up} = MV_{Uplayer} \quad (3.26)$$

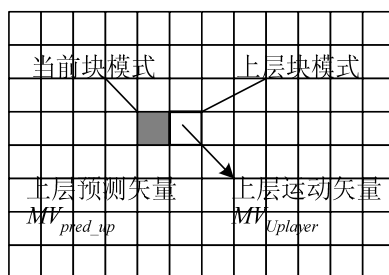


图 3.20 空间域上层预测的模式

## 2) 运动矢量在时间域预测方式

### a、前帧对应块运动矢量预测（Corresponding-block Prediction）

利用前一帧的与当前块相同坐标位置的块的运动矢量来作为当前块的预测运动矢量，如图 3.21 所示。

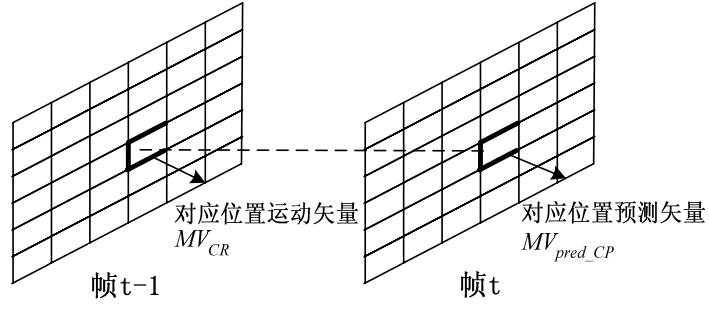


图 3.21 前帧对应位置的运动矢量预测的模式

b、时间域的邻近参考帧运动运动矢量预测（Neighboring Reference-frame Prediction）

由于视频序列的连续性，当前块在不同的参考帧中的运动矢量也是有一定的相干性的。如图 3.22 所示，假设当前块所在帧的时间为  $t$ ，则当在后面的参考帧  $t'$  中搜索当前块的最优匹配块时，可以利用当前块在参考帧  $t'+1$  中的运动矢量来估测出当前块在帧他  $t'$  的运动矢量：

$$MV_{pred\_NRP} = MV_{NR} \times (t - t') / (t' - t - 1) \quad (3.27)$$

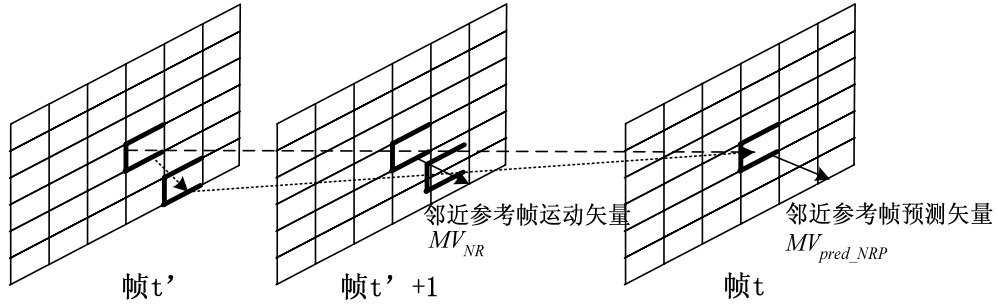


图 3.22 时间域邻近参考帧预测的模式

在上述运动矢量的四种预测方式中，经过实验证明，空间域的预测更为准确，其中上层块预测的性能最优，因为其充分利用了不同预测块模式运动矢量之间的相关性。而中值预测性能随着预测块尺寸的减小而增加，这是因为当前块尺寸越小，相关性越强。

#### 3.1.4.3.4 匹配误差在时空域的预测方式

H.264 中定义的匹配误差函数如下：

$$J(MV, \lambda_{MOTION}) = SAD(s, c(MV)) + \lambda_{MOTION} \times R(MV - PMV) \quad (3.28)$$

其中  $SAD$ （绝对差值和）计算公式如下：

$$SAD(s, c(\overline{MV})) = \sum_{x=1, y=1}^{B_x, B_y} |s[x, y] - c[x - MV_x, y - MV_y]|, B_x, B_y = 16, 8 \text{ or } 4 \quad (3.29)$$

其中  $s$  是当前进行编码的原始数据，而  $c$  是已经编码重建的用于进行运动补偿的参考帧的数据。 $MV$  为候选的运动矢量， $\lambda_{MOTION}$  为拉格朗日常数， $PMV$  为中值预测矢量， $R(MV - PMV)$  代表了运动矢量差分编码可能耗费的比特数，由于在接下来的四种匹配误差预测方式中匹配误差中的  $\lambda_{MOTION} \times R(MV - PMV)$  部分通常很接近而抵消， $SAD$  部分的预测特性基本上可以反映整个匹配函数的预测特性，因此  $J(MV, \lambda_{MOTION})$  用  $SAD$  来表示。

匹配误差在时空域的预测方式和运动矢量类似，具体分为：

## 1) 匹配误差在空间域预测方式

### a、中值预测 (Median Prediction)

与当前块 E 相邻的左边块 A，上边块 B 和右上方的块 C 搜索得到的最小 SAD 值分别为  $SAD_A$ ， $SAD_B$ ， $SAD_C$ ，取当前块的预测 SAD 值为：

$$SAD_{pred\_MD} = \min (SAD_{x\_median}, SAD_{y\_median}) \quad (3.30)$$

$SAD_{x\_median}$  是相邻块中对应运动矢量横坐标为

$$x\_median = \text{Median} (MV_A(x), MV_B(x), MV_C(x)) \quad (3.31)$$

的相邻块的最小 SAD 值， $SAD_{y\_median}$  也同理。如图 3.23 所示

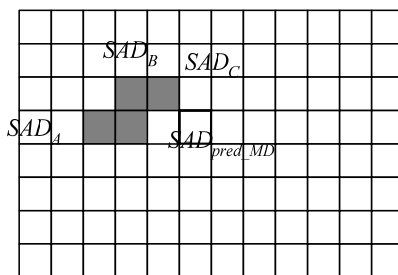


图 3.23 SAD 空间域中值预测模式

### b、上层预测 (Uplayer Prediction)

H.264 标准中提供的块尺寸有  $16 \times 16$ ， $8 \times 16$ ， $16 \times 8$ ， $8 \times 8$ ， $8 \times 4$ ， $4 \times 8$ ， $4 \times 4$ ，它们的图象分割区域分别定义为搜索模式  $Mode1$ - $Mode7$ 。假设当前分割区域的预测模式为  $Mode_{curr}$ ，上层模式  $Mode_{up}$  定义为

$$Mode_{up} = \begin{cases} \text{unavailable}, & \text{if } Mode_{curr} == Mode1 \\ Mode1, & \text{if } Mode_{curr} == Mode2, Mode3 \\ Mode2, & \text{if } Mode_{curr} == Mode4 \\ Mode4, & \text{if } Mode_{curr} == Mode5, Mode6 \\ Mode5, & \text{if } Mode_{curr} == Mode7 \end{cases} \quad (3.32)$$

利用空间域上层搜索模式的搜索所得的最小 SAD 值作为当前模式下的预测 SAD 值的方法如图 3.24 所示：

$$SAD_{pred\_up} = SAD_{uplayer}/2 \quad (3.33)$$

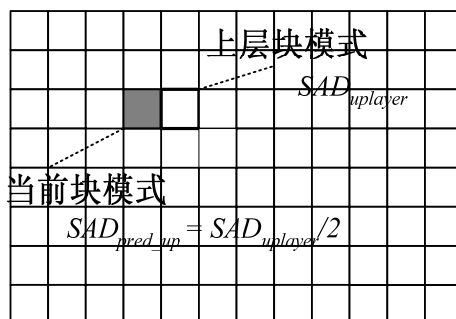


图 3.24 SAD 空间域上层预测模式

## 2) 匹配函数在时间域预测方式

### a、前帧对应块的预测 (Corresponding-block Prediction)

利用前一帧的与当前块相同坐标位置的块在帧  $t-1$  中搜索得到的最小 SAD，作为当前搜索块在

帧  $t'$  中进行搜索的  $SAD$  的预测值, 如图 3.25 所示:

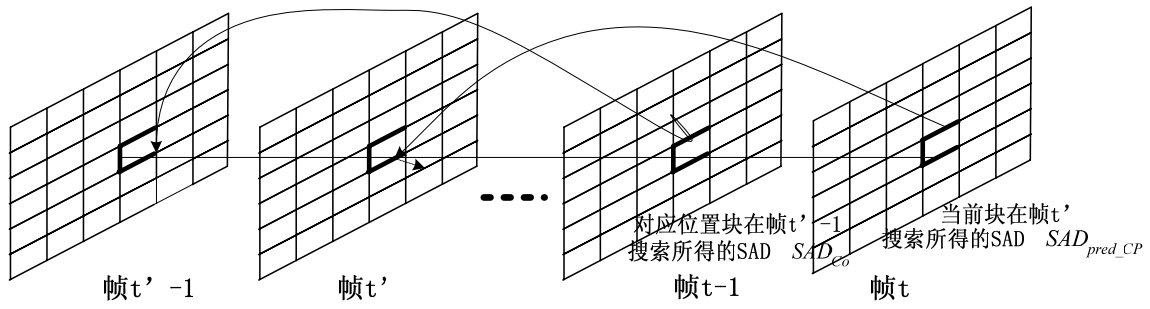


图 3.25 SAD 时间域前帧对应块预测模式

#### b、时间域的邻近参考帧预测 (Neighboring Reference-frame Prediction)

如图 3.26 所示, 假设当前块所在帧的时间为  $t$ , 则当在后面的参考帧  $t'$  中搜索时, 可以利用当前块在参考帧  $t'+1$  中搜索得到的最小  $SAD$  值  $SAD_{NR}$  作为当前块在帧  $t'$  搜索的  $SAD$  预测值, 即  $SAD_{pred\_NRP} = SAD_{NR}$ 。

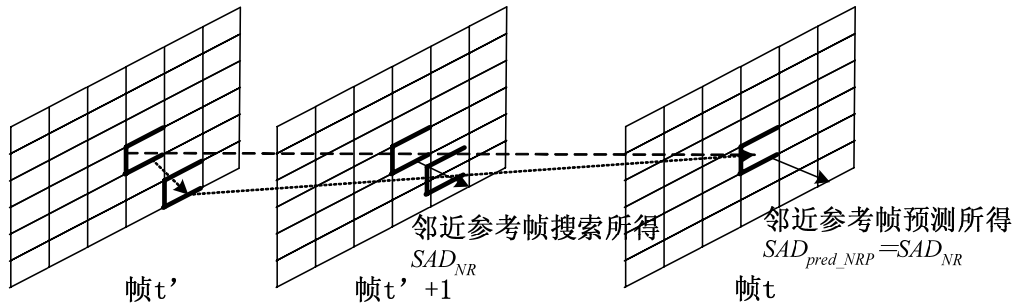


图 3.26 SAD 时间域邻近参考帧预测模式

以上所述的匹配误差的四种预测方式中, 时间域的预测更为准确, 其中邻近参考帧预测方式性能最优, 这是由于在这种预测方式下当前块相同, 邻近参考帧预测方式中, 邻近的参考帧之间具有比较强的相关性。但是它的性能受量化参数影响比较大, 这是由于量化参数大的时候图象细节模糊, 相邻参考帧图象内容之间的相似度增加, 因而用邻近参考帧预测的方式会更为准确一些。

#### 3.1.4.4 运动估计准则分类

运动搜索的目的就是在搜索窗内寻找与当前块最匹配的数据块, 这样就存在着如何判断两个块是否匹配的问题, 即如何定义一个匹配准则。而匹配准则的定义与运算复杂度和编码效率都是直接相关的, 通常有如下几类比较常用的匹配函数的定义:

设当前帧  $f_2$ , 参考帧  $f_1$ ,

##### (1) 最小均方差函数 (MSE)

$$MSE(MV) = \sum [f_2(x, MV) - f_1(x)]^2 \quad (3.34)$$

(2) 最小平均绝对值误差 (MAD) 等效于常用的绝对差值和 (SAD) 准则, 性能很好, 而且相对简单的硬件需求, 因而得到了最广泛的应用。

$$MAD(MV) = \sum [f_2(x, MV) - f_1(x)] \quad (3.35)$$

##### (3) 阈值差别计数 (NTD)

$$NTD(MV) = \sum G(f_2(x, MV) - f_1(x)) \quad (3.36)$$

其中:

$$\text{当 } |\alpha - \beta| > T_0 \text{ 时, } G(\alpha, \beta) = 1;$$

$$\text{当 } |\alpha - \beta| < T_0 \text{ 时, } G(\alpha, \beta) = 0 \quad (3.37)$$

由于在用块匹配算法进行运动估计的过程中，利用匹配准则函数进行匹配误差的计算是最主要的计算量，因此，我们可以从这方面进一步减少计算量。由于图象的帧内也具有相关性，在计算误差匹配函数时，可以只让图象块中的部分像素参与运算，将块中的所有像素组成一个集合，那么参与计算的这部分像素集合就是它的子集，这种误差匹配的方法被称为子集匹配法。实验结果表明，在匹配误差无明显增加的情况下，采用子集匹配可以大大减少每帧图象的平均搜索时间。

### 3.1.4.5 运动搜索算法

匹配误差函数，可以用各种优化方法进行最小化，这就需要我们开发出高效的运动搜索算法，主要的几种算法归纳如下：

#### 3.1.4.5.1 全局搜索算法

为当前帧的一个给定块确定最优位移矢量的全局搜索算法方法是：在一个预先定义的搜索区域内，把它与参考帧中所有的候选块进行比较，并且寻找具有最小匹配误差的一个。这两个块之间的位移就是所估计的 MV，这样做带来的结果必然导致极大的计算量。

选择搜索区域一般是关于当前块对称的，左边和右边各有  $R_x$  个像素，上边和下边各有  $R_y$  个像素，如图 3.27 所示。

如果已知在水平和垂直方向运动的动态范围是相同的，那么  $R_x = R_y = R$ 。估计的精度是由搜索的步长决定的，步长是相邻两个候选块在水平或者垂直方向上的距离。通常，沿着两个方向使用相同的步长。在最简单的情况下，步长是一个像素，称为整数像素精度搜索，该种算法也称为无损搜索算法。

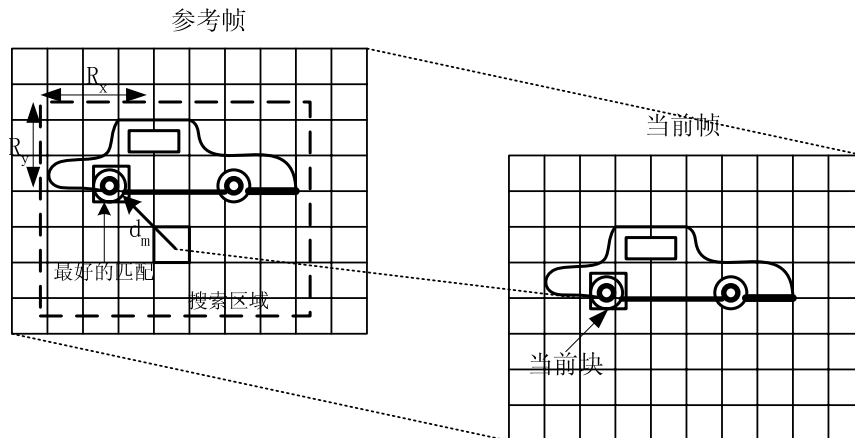


图 3.27 全局搜索算法的搜索过程

#### 3.1.4.5.2 分数精度搜索算法

由于在穷尽块匹配算法中搜索相应块的步长不一定是整数，一般来说，为了实现  $1/K$  像素步长，对参考帧必须进行  $K$  倍内插。图 3.28 给出了  $K=2$  的例子，它被称为半像素精度搜索。根据实验证明，与整像素精度搜索相比，半像素精度搜索在估计精度上有很大提高，特别是对于低清晰度视频。



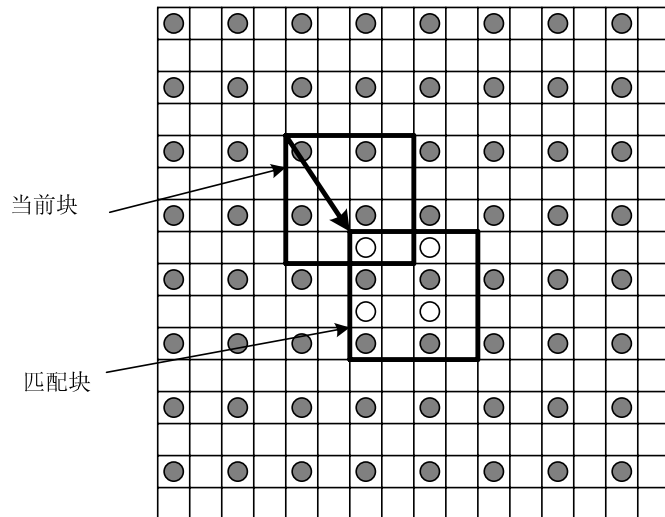


图 3.28 半像素精度块匹配

但是，应用分数像素步长，搜索算法的复杂性大大增加，例如，使用半像素搜索，搜索点的总数比整数像素精度搜索大四倍以上。

那么，如何确定适合运动估计的搜索步长，对于视频编码的帧间编码来说，即使得预测误差最小化。预测误差和搜索精度之间的关系的统计分析如图 3.29 所示：

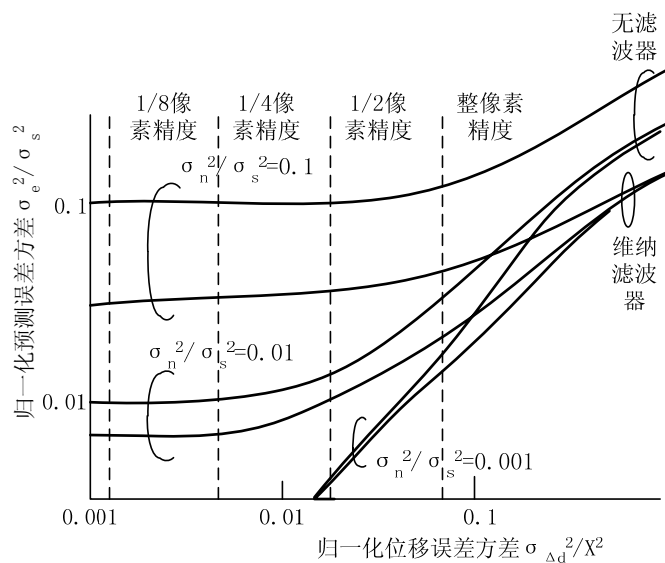


图 3.29 运动补偿精度对有噪声信号的预测误差方差的影响

从图中可以看出，在低噪声情况下，需要不大于 1/8 像素的精度，而在高噪声情况下，1/2 像素精度就够了。

#### 3.1.4.5.3 快速搜索算法

快速搜索算法和全局搜索算法相比，虽然只能得到次最佳的匹配结果，但在减少运算量方面效果显著。

##### 1) 二维对数搜索法

这种算法的基本思路是采用大菱形搜索模式和小菱形搜索模式，步骤如图 6.4.20 所示，从相应于零位移的位置开始搜索，每一步试验菱形排列的五个搜索点。下一步，把中心移到前一步找到的

最佳匹配点并重复菱形搜索。当最佳匹配点是中心点或是在最大搜索区域的边界上时，就减小搜索步长（菱形的半径）。否则步长保持不变。当步长减小到一个像素时就到达了最后一步，并且在这最后一步检验九个搜索点。初始搜索步长一般设为最大搜索区域的一半。

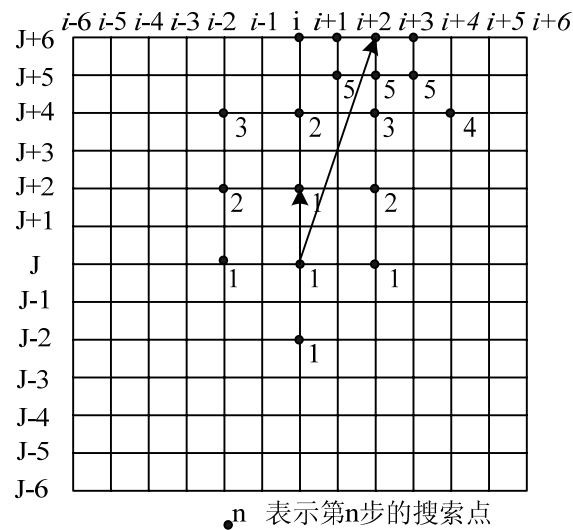


图 3.30 二维对数搜索法

其后这类算法在搜索模式上又做了比较多的改进，在搜索模式上采用了矩形模式，还有六边形模式、十字形模式等等。

## 2) 三步搜索法

如图 3.31 所示，这种搜索的步长从等于或者略大于最大搜索范围的一半开始。第一步，在起始点和周围八个“1”标出的点上计算匹配误差，如果最小匹配误差在起始点出现，则认为没有运动；第二步，以第一步中匹配误差最小的点（图中起始点箭头指向的“1”）为中心，计算以“2”标出的 8 个点处的匹配误差。注意，在每一步中搜索步长都比上一步长减少一半，以得到更准确的估计；在第三步以后就能得到最终的估计结果，这时从搜索点到中心点的距离为一个像素。

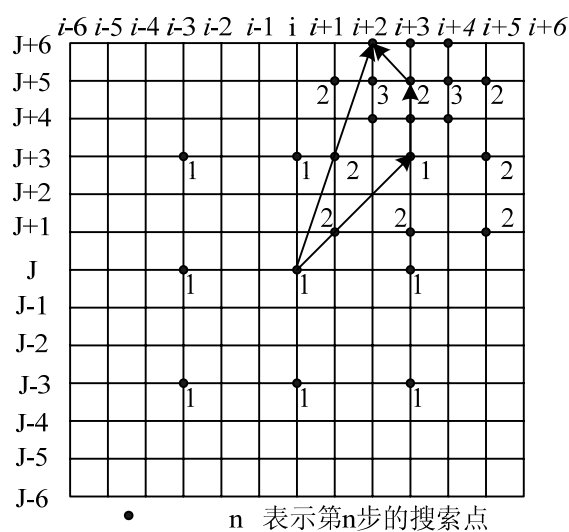


图 3.31 三步搜索法

此外，在[6]中还提到采用了钻石搜索模型（Diamond search strategy）。

但是，上述一些快速算法更适合用于估计运动幅度比较大的场合，对于部分运动幅度小的场合，它们容易落入局部最小值而导致匹配精度很差，已经有很多各种各样的视频流证明了这一点。

现在，针对这一缺点，国内外诸多专家学者也提出了相应的应对措施，特别是针对H.264编码标准要求的一些快速算法的改进，并取得卓越的效果。例如[7]中提到的基于全局最小值具有自适应性的快速算法，这种算法通过在每一搜索步骤选择多个搜索结果，基于这些搜索结果之间的匹配误差的不同得到的最佳搜索点，因而可以很好地解决落入局部最小值的问题。

[8]中提到一种适用于H. 264的基于自适应搜索范围的快速运动估计算法，经过实验证明对于如salesman等中小运动序列，其速度可接近全局搜索算法的400倍，接近三步搜索算法的4倍；而对于大运动序列，如table tennis，该算法则会自动调节搜索点数以适应复杂的运动。当从总体上考察速度方面的性能时，可以看到，该算法平均速度是全局搜索算法的287.4倍，三步搜索的2.8倍。

#### 3.1.4.5.4 分级搜索范围（DSR）算法

分级搜索算法的基本思想是从最低分辨率开始逐级精度的进行不断优化的运动搜索策略，如图3.32所示，首先取得两个原始图象帧的金字塔表示，从上到下分辨率逐级变细，从顶端开始，选择一个尺寸比较大的数据块进行一个比较粗略的运动搜索过程，对在此基础上进行亚抽样（即通过降低数据块尺寸（或提高抽样分辨率）和减少搜索范围的办法）进行到下一个较细的级来细化运动矢量，而一个新的搜索过程可以在上一级搜索到的最优运动矢量周围进行。在亚抽样的过程中也有着不同的抽样方式和抽样滤波器。这种方法的优点是运算量的下降比例比较大，而且搜索的比较全面。缺点是由于亚抽样或者滤波器的采用而使内存的需求增加，另外如果场景细节过多可能会容易落入局部最小点。

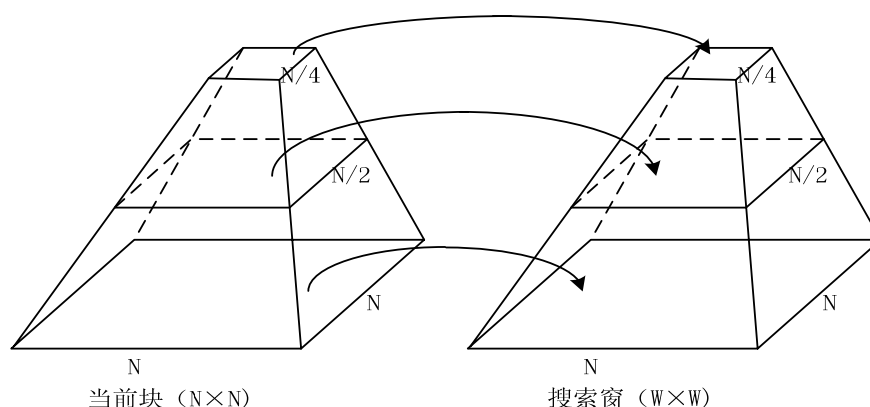


图 3.32 分级运动搜索过程示意图

#### 3.1.4.5.5 混合搜索算法

由于物体的运动千变万化，很难用一种简单的模型去描述，也很难用一种单一的算法来搜索最佳运动矢量，因此实际上大多采用多种搜索算法相组合的办法，可以在很大程度上提高预测的有效性和鲁棒性，举例说明如图3.33为

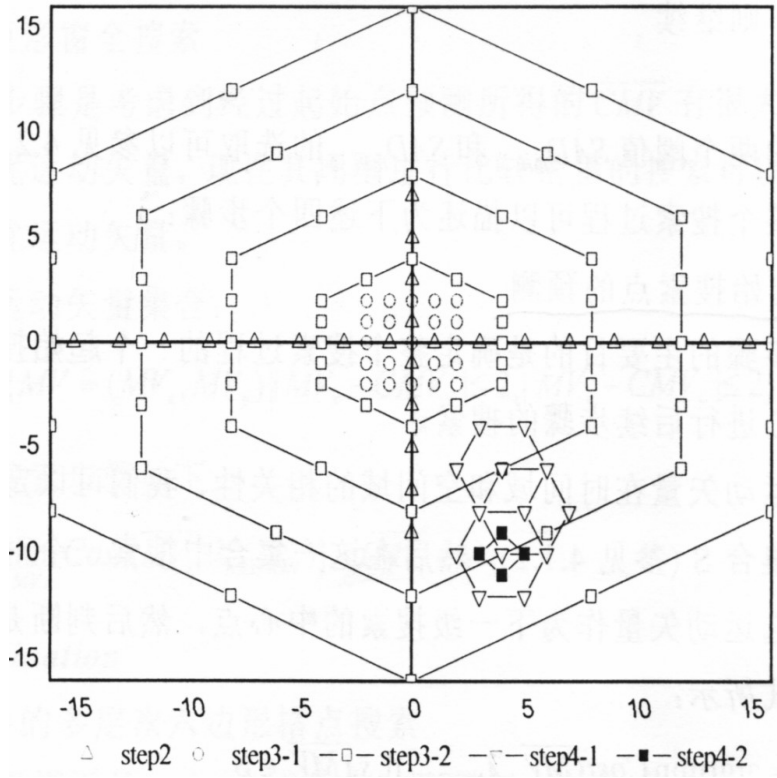


图 3.33 非对称十字型多层次六边形格点运动搜索算法

具体搜索方法的步骤如下：

**Step1:** 搜索过程的起始搜索点搜索，

在起始搜索矢量的集合  $S$  中搜索一个对应费用函数值最小的候选运动矢量作为起始搜索点，费用函数表达式如下：

$$\mathbf{m}_{\min} = \arg[\min_{\mathbf{m}_i} J(\mathbf{m}_i, \lambda_{MOTION})], s.t. \mathbf{m}_i \in S$$

然后判断是否提前截止 **Early\_Termination**。

**Step2:** 不对称的十字交叉搜索

由于自然界物体运动水平方向要比垂直方向剧烈一些，采用非对称的十字形搜索方法，搜索点集合为：

$$\Omega_1 = \{\mathbf{m} = (m_x, m_y)^T \mid \mathbf{m} = (cm_x \pm 2i, cm_y)^T, i = 0, 1, 2, \dots, \frac{W}{2}; \mathbf{m} = (cm_x, cm_y \pm 2j)^T, j = 1, 2, \dots, \frac{W}{4}\} \quad \mathbf{cm} = \mathbf{m}_{\min}$$

$W$  为搜索区。

搜索步骤如下：

$$(1) \quad \mathbf{m}_{\min 2} = \arg[\min_{\mathbf{m}_i} J(\mathbf{m}_i, \lambda_{MOTION})], s.t. \mathbf{m}_i \in \Omega_1$$

$$(2) \quad \mathbf{m}_{\min} = \arg[\min(J(\mathbf{m}_{\min}, \lambda_{MOTION}), J(\mathbf{m}_{\min 2}, \lambda_{MOTION}))]$$

(3) **Early\_Termination**

**Step3:** 非均匀多层次六边形格点搜索，分成两个子步骤：

**Step3-1:** 小矩形窗全搜索

定义候选运动矢量集合：

$$\Omega_2 = \{\mathbf{m} = (m_x, m_y)^T \mid |m_x - cm_x| \leq 2, |m_y - cm_y| \leq 2\}, \quad \mathbf{cm} = \mathbf{m}_{\min}$$

然后搜索步骤有：

$$\mathbf{m}_{\min 3} = \arg[\min_{\mathbf{m}_i} J(\mathbf{m}_i, \lambda_{MOTION})], s.t. \mathbf{m}_i \in \Omega_2$$

$$\mathbf{m}_{\min} = \arg[\min(J(\mathbf{m}_{\min}, \lambda_{MOTION}), J(\mathbf{m}_{\min 3}, \lambda_{MOTION}))]$$

### Early\_Termination

**Step3-2:** 扩展的多层次六边形格点搜索

六边形的 16 个点为：

$$\Omega_{16-HP} = \{\mathbf{m} = (x, y)^T \mid \mathbf{m} = (\pm 4, \pm 2)^T, (\pm 4, \pm 1)^T, (\pm 4, 0)^T, (\pm 2, \pm 3)^T, (0, \pm 4)^T\}$$

采用下列办法扩展搜索区：

for k= 1 k< W/4 k++

{

$$\Pi_k = \{\mathbf{m} = (m_x, m_y) \mid m_x = cm_x + k \cdot x', m_y = cm_y + k \cdot y', (x', y') \in \Omega_{16-HP}\}, \text{ 然后有:}$$

$$\mathbf{m}_{\min k} = \arg[\min_{\mathbf{m}_i} J(\mathbf{m}_i, \lambda_{MOTION})], s.t. \mathbf{m}_i \in \Pi_k$$

$$\mathbf{m}_{\min} = \arg[\min(J(\mathbf{m}_{\min}, \lambda_{MOTION}), J(\mathbf{m}_{\min k}, \lambda_{MOTION}))]$$

**Early\_Termination;**

}

**Step4:** 扩展的六边形搜索，分成两个子步：

**Step4-1:** 六边形搜索模型定义为：

$$\Omega_3 = \{\mathbf{m} = (m_x, m_y)^T \mid \mathbf{m} = (cm_x \pm 2, cm_y)^T, (cm_x \pm 1, cm_y \pm 2)^T\}, \quad \mathbf{cm} = \mathbf{m}_{\min}$$

然后有

$$\mathbf{m}_{\min 4} = \arg[\min_{\mathbf{m}_i} J(\mathbf{m}_i, \lambda_{MOTION})], s.t. \mathbf{m}_i \in \Omega_3$$

$$\mathbf{m}_{\min} = \arg[\min(J(\mathbf{m}_{\min}, \lambda_{MOTION}), J(\mathbf{m}_{\min 4}, \lambda_{MOTION}))]$$

如果  $\mathbf{m}_{\min} == \mathbf{cm}$ ，跳转至 **Step4-2**;

否则，跳转至 **Step4-1**。

**Step4-2:** 基于菱形模块的搜索

这一步骤的基本搜索模块为图 6.4.25 中所示的菱形模块，其集合为

$$\Omega_4 = \left\{ m = (m_x, m_y)^T \mid m = (cm_x \pm 1, cm_y)^T, (cm_x, cm_y \pm 1)^T \right\}, \quad \mathbf{cm} = \mathbf{m}_{\min}$$

然后开始搜索步骤如下：

$$m = \arg[\min Cost(m, \lambda_{MOTION})], m \in \Omega_4$$

如果  $\mathbf{m}_{\min} == \mathbf{cm}$ ，搜索截止

否则，跳转至 Step4-2。

事实上，在运动估计时也并不是单一使用上述某一类搜索算法，而是根据各类算法的优点灵活组合采纳。在运动幅度比较大的情况下可以采用自适应的菱形搜索法和六边形搜索法，这样可以大大节省码率而图象质量并未有所下降。在运动图象非常复杂的情况下，采用全局搜索法在比特数相对来说增加不多的情况下使得图象质量得到保证。H.264 编码标准草案推荐使用 1/4 分数像素精度搜索。[6]中提到在整像素搜索时采用非对称十字型多层次六边形格点运动搜索算法，然后采用钻石搜索模型来进行分数像素精度运动估计。

解码器要求传送的比特数最小化，而复杂的模型需要更多的比特数来传输运动矢量，而且易受噪声影响。因此，在提高视频的编码效率的技术中，运动补偿精度的提高和比特数最小化是相互矛盾的，这就需要在运动估计的准确性和表示运动所用的比特数之间作出折中的选择。它的效果与选用的运动模型是密切相关的。

## 3.2 变换编码

### 3.2.1 变换编码的基本概念

绝大多数图像都有一个共同的特征：平坦区域和内容缓慢变化区域占据一幅图像的大部分，而细节区域和内容突变区域则占小部分。也可以说，图像中直流和低频区占大部分，高频区占小部分。这样，空间域的图像变换到频域或所谓的变换域，会产生相关性很小的一些变换系数，并可对其进行压缩编码，即所谓的变换编码。

变换中有一类叫做正交变换，可用于图像编码。自 1968 年利用快速傅立叶变换（FFT）进行图像编码以来，出现了多种正交变换编码方法，如 **K-L 变换**、**离散余弦变换（DCT）** 等等。其中，编码性能以 **K-L 变换最理想**，但缺乏快速算法，且变换矩阵随图像而异，不同图像需计算不同的变换矩阵，因而只用来参考比较。DCT 编码性能最接近于 K-L 变换，略次而已，具有快速算法，广泛应用于图像编码。

### 3.2.2 K-L 变换

设一图像阵列每行由列矢量表示，即  $\mathbf{x} = (x_0, x_1, \dots, x_{N-1})^T$ 。由于图像内容的千变万化，该列矢量是随机矢量，用其平均值和协方差表征。

$\mathbf{X}$  的列矢量平均值  $\bar{\mathbf{X}}$  为：

$$\bar{\mathbf{X}} = E[\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{N-1}]^T = [\bar{x}_0, \bar{x}_1, \dots, \bar{x}_{N-1}]^T \quad (3.42)$$

$\mathbf{X}$  的协方差矩阵定义为：

$$C_x = E\{(X-\bar{X})(X-\bar{X})^T\} = \begin{pmatrix} c_{00} & \cdots & c_{0,N-1} \\ \vdots & \ddots & \vdots \\ c_{N-1,0} & \cdots & c_{N-1,N-1} \end{pmatrix} \quad (3.43)$$

上式中,  $c_{ij} = E\{(x_i - \bar{x}_i)(x_j - \bar{x}_j)\}$ , 表示随机变量  $x_i$  和  $x_j$  之间的相关程度的协方差 (相关系数)。协方差矩阵对角线上的元素是各像素的方差  $c_{ii} = \sigma_i^2$ , 表示个像素的能量大小。

不难求出  $X$  的协方差矩阵  $C_x$  的特征值及其归一化特征矢量。利用这些特征矢量构成正交矩阵  $A$ , 则图像  $X$  作正交变换后得:

$$Y = AX \quad (3.44)$$

该变换便是 K-L 变换, 也叫做 Hotelling 变换。可以证明,  $Y$  得协方差矩阵是一个对角矩阵, 其对角线上元素便是  $C_x$  得特征值。

$$C_Y = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_N \end{pmatrix} \quad (3.45)$$

这说明 K-L 变换后,  $Y$  中的相关性已全部消除, 能量只集中在  $n$  个特征值  $\lambda_i$  上, 其值按大到小排列。因此编码只需传送前  $n$  个特征值, 大大降低码率。可以证明, 在均方误差最小准则下, K-L 变换是失真最小的变换。但 K-L 变换不易实现, 只是理想的变换。

### 3.2.3 离散余弦变换 DCT

#### 3.2.3.1 一维 DCT 变换

DCT 正变换时, 设  $f(x) = [f(0), f(1), \dots, f(N-1)]^T$ , 对  $f$  变换后得:

$$F(u) = \sqrt{\frac{2}{N}} c(u) \sum_{x=0}^{N-1} f(x) \cos \left[ \frac{\pi}{2N} (2x+1)u \right] \quad (3.46)$$

$$u = 0, 1, \dots, N-1$$

其中,  $c(u) = \begin{cases} 1/\sqrt{2}, u=0 \\ 1, u=1, 2, \dots, N-1 \end{cases}$

可写成矩阵形式,

$$F = AF \quad (2.6)$$

$$A = \sqrt{2/N} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \dots & 1/\sqrt{2} \\ \cos \frac{\pi}{2N} & & \\ \vdots & \ddots & \vdots \\ \cos \frac{(N-1)\pi}{2N} & \dots & \cos \frac{(2N-1)(N-1)\pi}{2N} \end{pmatrix} \quad (3.47)$$

对应地，一维离散余弦反变换 IDCT 为：

$$f(x) = \sqrt{\frac{2}{N}} \sum_{u=0}^{N-1} c(u) F(u) \cos \left[ \frac{\pi}{2N} (2x+1)u \right] \quad x=0,1,2,\dots,N \quad (3.48)$$

由于  $A^{-1}=A^T$ ， $f$  可写成如下矩阵形式：

$$f = A^{-1}F = A^T F \quad (3.49)$$

### 3.2.3.2 二维 DCT 变换

令  $f(x, y)$  为  $N \times N$  离散图像序列，二维 DCT 变换表示为：

$$F(u, v) = \frac{2}{N} c(u) c(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos \left[ \frac{\pi}{2N} (2x+1)u \right] \cos \left[ \frac{\pi}{2N} (2y+1)v \right] \quad (3.50)$$

$$u, v = 0, 1, \dots, N-1$$

二维 IDCT 为：

$$f(x, y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} c(u) c(v) F(u, v) \cos \left[ \frac{\pi}{2N} (2x+1)u \right] \cos \left[ \frac{\pi}{2N} (2y+1)v \right] \quad (3.51)$$

$$x, y = 0, 1, \dots, N-1$$

也可写成矩阵形式：

$$F = AfA^T \quad (3.52)$$

$$f = A^T F A \quad (3.53)$$

对于图像变换编码，最理想的变换操作应对整个图像进行，以便去除所有像素间的相关性。但这样的操作计算量太大。实际上，往往把图像分为若干块，以块为单位进行 DCT 变换。通常区  $16 \times 16$  或  $8 \times 8$  组成小块。

图像块通常只需用几个低频 DCT 系数表示，如图 3.33 所示。



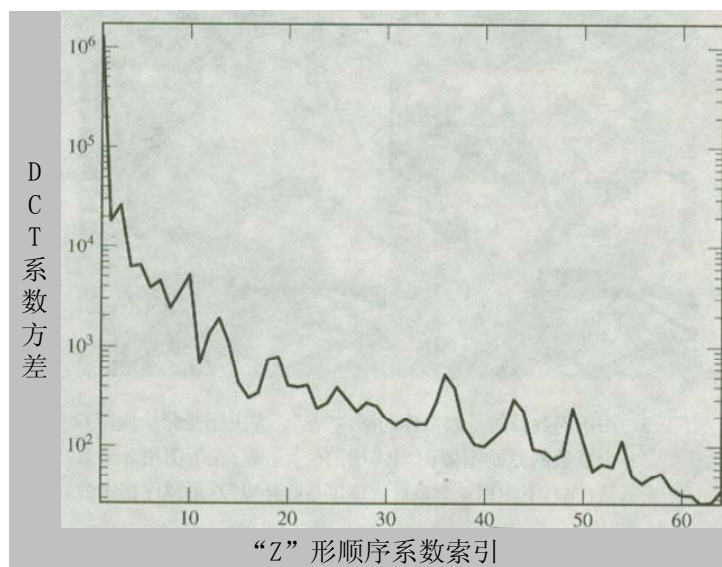


图 3.33 8×8DCT 能量分布

由图 3.34 可见，每块用 64 个系数表示的图像，变换后只需 16 个系数表示原始图像。

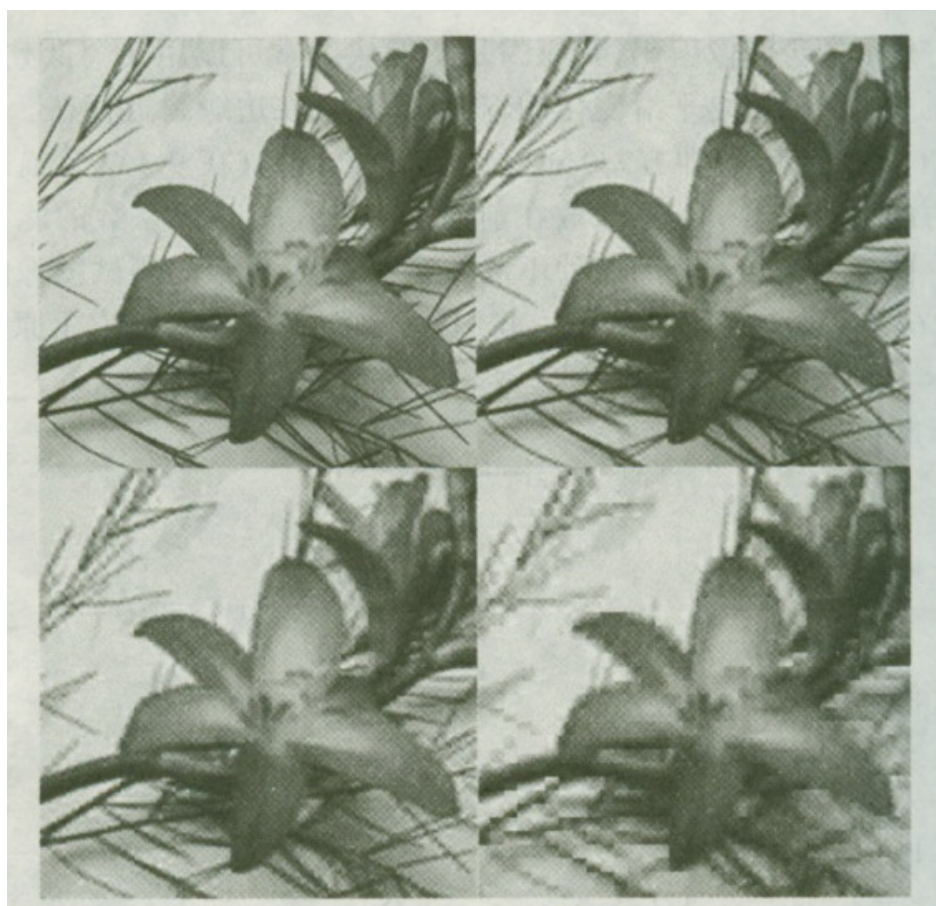


图 3.34 不同数目系数重建图像。

左上图 63 各系数，右上图 16 个系数，左下图 8 个系数，右下图 4 个系数

### 3.2.4 锯齿形扫描和游程编码

变换系数量化后，在低频和直流区域少量较大的值，高频区域由少量不大的值，系数大部分为零，为了更有效的编码，通常根据该统计特性采用熵编码进一步压缩码率。

两者各有优缺点，特别是变换编码随着 VLSI 技术的飞跃发展，实现起来十分容易。现实中，往往采用混合编码方法，即对图像先进行带有运动补偿的帧间预测编码，再对预测后残差信号进行 DCT 变换。这种混合编码方法已成为许多视频压缩编码国际标准的基本框架。

3.4 熵编码

利用信源的统计特性进行码率压缩的编码就称为熵编码，也叫统计编码。视频编码常用的有两种：变长编码，也称哈夫曼编码及算术编码。现分别讨论其基本原理。

3.4.1 变长编码

1952 年，哈夫曼提出变长编码方法：对出现概率大的符号分配短字长的二进制码，对出现概率小的符号分配长字长二进制码，得到符号平均码长最短的码。变长编码也称最佳编码。具体编码方法如图 3.36 所示例子。

例：

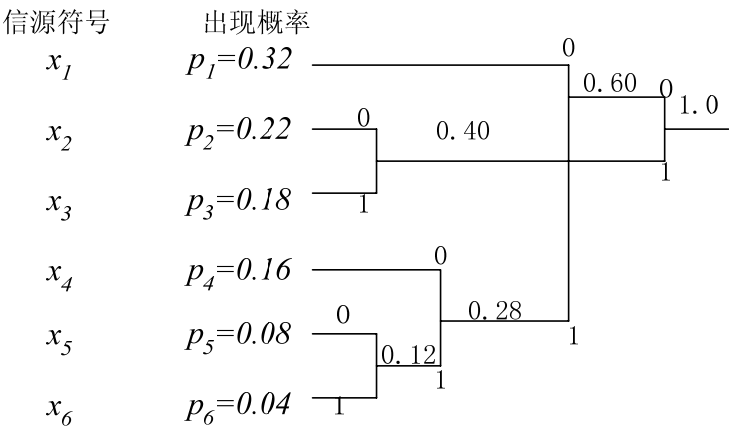


图 3.36 哈夫曼编码举例

可得：

表 3.2 结果

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$
$l_i$	2	2	2	3	4	4
$w_i$	00	10	11	010	0110	0111

则平均码长  $\bar{w} = \sum_{i=1}^6 p_i l_i = 2.32$ 。

哈夫曼编码步骤如下：

第一步，将信息符号按其出现概率从大到小排列；

第二步，将两个最小概率组成一组，划成 2 个分支域，并标以 0 和 1；再把 2 个分支域合并成 1 个支域，标以两个概率之和；

第三步，依次类推，直到概率之和等于 1.0；

第四步，找出概率和 1.0 到各信息符号的路径，记下各路径从右到左各分支域的 0 和 1，即得到信息符号相应的码字。

理论上，这种编码是最佳的。实际上，利用硬件实现时，出现概率不可能精确到小数后多少位，而最小存储单元为 1bit，会引起概率匹配不准确及编码效率的下降。

3.4.2 算术编码

算术编码和哈夫曼编码不同，不采用一个码字代表一个输入信息符号的办法，而采用一个浮点数来代替一串输入符号，经算术编码后输出一个小于 1，大于或等于 0 的浮点数，在解码端被正确地唯一的解码，恢复原符号序列。举例如下。

设：输入序列为 abaca， $p(a)=1/2$ ， $p(b)=1/2=p(c)=1/2$ ，求算术编码输出序列。

解：

编码：

(1) 列出各符号出现概率值

表 3.3 符号出现概率

字符	概率	范围
a	0.5	[0.00,0.50)
b	0.25	[0.50,0.75)
c	0.25	[0.75,1.00)

(2) 在 (0, 1) 区间内，每个字符根据其概率选定范围，如表所示

(3) 开始时，浮点范围： $R_0=H_0-L_0=1$ ， $H_0=1$ ， $L_0=0$ 。

(4) 当第一个字符“a”被传送时，其范围为[0.00,0.50)， $H=0.05$ ， $L=0.00$ ，可得发 a 字符的范围 H 和 L：

$$L_1=L_0+R_0\times L=0.00$$

$$H_1=L_0+R_0\times H=0.50$$

$$\text{范围 } R_1=H_1-L_1=0.50$$

对 a 编码后，编码范围从[0, 1) 变为[0.00,0.50)。

(5) 当第二个字符“b”被传送时，范围为[0.50,0.75)， $H=0.75$ ， $L=0.50$ 。

$$L_2=L_1+R_1\times L=0.25$$

$$H_2=L_1+R_1\times H=0.375$$

$$R_2=H_2-L_2=0.125$$

于是对“ab”编码后，编码范围从[0.00,0.50)变为[0.25,0.375)。

(6) 依次类推：

$$\text{“a”： } L_3=0.25+0.125\times 0=0.25$$

$$H_3=0.25+0.125\times 0.50=0.3125$$

$$R_3=0.0625$$

$$\text{“c”： } L_4=0.25+0.0625\times 0.75=0.296875$$

$$H_4=0.25+0.0625\times 1.00=0.3125$$

$$R_4=0.015625$$

$$\text{“a”： } L_5=0.296875+0.015625\times 0=0.296875$$

$$H_5=0.296875+0.015625\times 0.50=0.3046875$$

$$R_5=0.0078125$$

最后输出码字为：0.3046875。

表 3.4 算术编码结果

序列	范围 L	范围 H
初始	0	1
a	0.00	0.50
b	0.25	0.375
a	0.25	0.3125
c	0.296875	0.3125
a	0.296875	0.3046875

解码：

(1) 接受到浮点数 0.3046875，对照表 1，在范围中查得第一个字符为“a”，其概率为 0.5。

(2) 从接收值减去“a”得的概率范围 L，并除以  $p(a)$ ，

$$(0.3046875 - 0.00) / 0.5 = 0.609375$$

该值为下一字符范围内的值。

(3) 依此类推，解码出序列“abaca”。

表 3.5 算术解码结果

解码浮点数	输出字符	概率范围 L	概率范围 H	P
0.3046875	a	0.00	0.50	0.5
0.609375	b	0.50	0.75	0.25
0.4375	a	0.00	0.50	0.5
0.875	c	0.75	1.00	0.25
0.5	a	0.00	0.50	0.5

## 参考文献

1. 毕厚杰, 多媒体信息的传输与处理, 人民邮电出版社, 1999
2. 许志祥, 数字电视与图像通信, 上海大学出版社, 2000
3. Yao Wang, Jorn Ostermann, Ya-Qin Zhang, Video Processing and Communication, Pearson Education, 2002
4. Iain E.G.Richardson, H.264 and MPEG-4 Video Compression Video Coding for Next Generation Multimedia, Wiley Press , 2003
5. Lloyd, P.S. , Least square quantization in PCM, IEEE Trans. Information Theory, March 1982, p127-135
6. Orchard M.T., and G.J.Sullivan, OBMC, an estimation-theoretic approach, IEEE Trans Image Process(1994,3), P693-699
7. 陈志波 . H.264运动估值与网络视频传输关键问题研究, 清华大学, 毕业论文, 2002.
8. K. Venkatachalapathy, R. Krishnamoorthu, and K. Viswanath. A new adaptive search strategy for fast block motion estimation algorithms. Journal of Visual Communication & Image Representation, 2004: 203-213.
9. 李翔、吴国威.一种适用于H. 264的基于自适应搜索范围的快速运动估计算法 . 中国图象图形学报, 2004.4:471-476.

## 第4章 视频编码标准简介

### 4.1 视频编码发展简史

1984年CCITT第15研究组发布了数字基群电视会议编码标准H.120建议。1988年CCITT通过了“ $p \times 64\text{Kbps}$  ( $p=1,2,3,4,5,\dots,30$ )”视像编码标准H.261建议，被称为视频压缩编码的一个里程碑。从此，ITU-T、ISO等公布的基于波形的一系列视频编码标准的编码方法都是基于H.261中的混合编码方法。

1986年，ISO和CCITT成立了联合图像专家组(JPEG, Joint Photographic Experts Group)，研究连续色调静止图像压缩算法国际标准，1992年7月通过了JPEG标准。

1988年ISO/IEC信息技术联合委员会成立了活动图像专家组(MPEG, Moving Picture Expert Group)。1991年公布了MPEG-1视频编码标准，码率为1.5Mbps，主要应用于家用VCD的视频压缩；1994年11月，公布了MPEG-2标准，用于数字视频广播(DVB)、家用DVD的视频压缩及高清晰度电视(HDTV)。码率从4Mbps、15Mbps……直至100Mbps分别用于不同档次和不同级别的视频压缩中。

1995年，ITU-T推出H.263标准，用于低于64Kbps的低码率视频传输，如PSTN信道中可视会议、多媒体通信等。1984年和2000年又分别公布了H.263+、H.263++等标准。

1999年12月份，ISO/IEC通过了“视听对象的编码标准”——MPEG4，它除了定义视频压缩编码标准外，还强调了多媒体通信的交互性和灵活性。

2003年3月，ITU-T和ISO/IEC正式公布了H.264视频压缩标准，不仅显著提高了压缩比，而且具有良好的网络亲和性，加强了对IP网、移动网的误码和丢包的处理。有人将H.264称为新一代的视频编码标准。本书也将重点介绍该标准。

### 4.2 H.261 标准

由于会议电视和可视电话的需要，CCITT发布了码率为 $p \times 64\text{Kbps}$  ( $p=1,2,3,4,5,\dots,30$ )的H.261建议，这个视频编码方案对以后各种视频编码标准产生了深远影响，直至今日。

#### 4.2.1 图像格式

H.261用于视频通信，会产生多个国家的互通困难的问题，不同国家采用不同的彩电制式，不可能直接互通。H.261采用一种公共中间格式(CIF, Common Intermediate Format)，不论何种彩色格式，发送方先把自己国家的彩电制式转换成CIF格式，经H.261编码后再由CIF格式转换到接收方彩电制式，如表4.1所示

表4.1 CIF图像格式

采用的标准		ISO/MPEG-1		CCITT/H.261		CCIR601	
参数		SIF		CIF	QCIF	PAL	NTSC
每秒帧数		25	30	29.97		25	
每帧有效行数	Y	288	240	288	144	576	480
	Cr	144	120	144	72	288	240
	Cb	144	120	144	72	288	240
每行有效像素	Y	360	352	352	176	720	
	Cr	180	176	176	88	360	
	Cb	180	176	176	88	360	

为了比较，表4.1列了CCIR601规定的两种常用PAL和NTSC彩电制式的图像格式，

它并不属于 CIF 格式。表 1 中 SIF（Source Input Format）格式用于 MPEG-1。

采用 CIF 及 QCIF 格式时，视频信号的结构采用图 4.1 所示的图像、块组（GOB，group of block）、宏块（MB，macroblock）、块（B，block）四级结构。每帧 CIF 图像由 2 个 GOB 组成，每个 GOB 由 33 个 MB 组成，每个 MB 由 4 个亮度块和 1 个 Cr 块及 1 个 Cb 块组成，每个块（B）又由 8×8 像素构成。一帧 QCIF 图像由 3 个 GOB 组成。

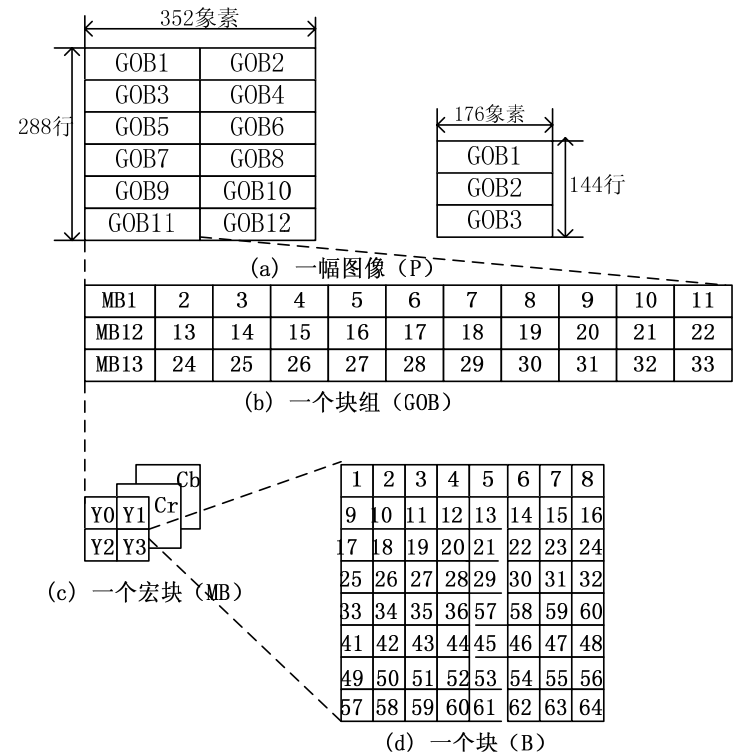


图 4.1 CIF 及 QCIF 结构

不论 CIF 还是 QCIF 格式帧都为 29.97 帧/秒≈30 帧/秒，如果每像素的量化取 8 比特，彩色格式取 4:2:0，码率分别为：

$$\text{CIF: } 352 \times 288 \times 8 \times 30 \times 1.5 = 36.5 \text{ Mbps}$$

$$\text{QCIF: } 176 \times 144 \times 8 \times 30 \times 1.5 = 9.1 \text{ Mbps}$$

显然，这样高的码率不经过视频压缩编码要通过 ISDN 的 64Kbps~2.048Mbps 信道是不可能的。

## 4.2.2 H.261 视频编解码器

### 4.2.3.1 H.261 编解码器系统

H.261 视频编解码器结构如图 4.2 所示，视频信源编码器用于视频信号的码率压缩，主要采用混合编码方法；视频复合编码器将每帧图像数据编排成四层结构，并通过熵编码对视频数据进一步压缩输出。传输缓冲器和码率控制器用于保证输出码流尽可能稳定。传输编码器则用于视频数据的误码检测和纠正。解码器各部分功能与编码器相反。



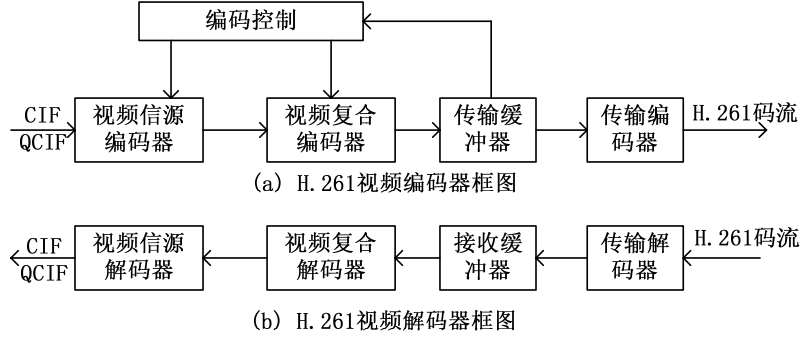


图 4.2 H.261 视频编码器框图

#### 4.2.2.2 视频信源编码器

视频信源编码器结构框图见图 4.3 所示，输入以 MB 为单位。

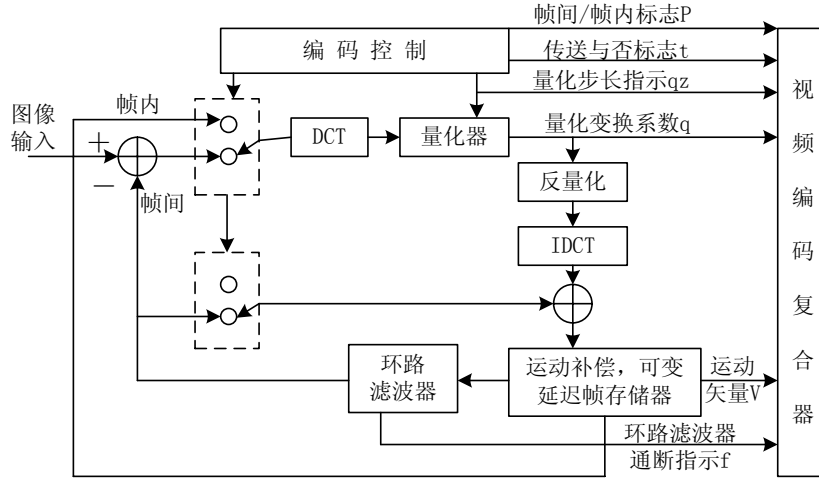


图 4.3 H.261 信源编解码器

##### (1) 帧间编码和帧内编码自动判决准则

一般可根据帧间相关性判定。相关性大则采用帧间编码，否则用帧内编码。帧间相关性可用当前帧与过去帧的对应像素差的均方值表示：

$$\text{VAR} = \frac{1}{256} \sum_{x=1}^{16} \sum_{y=1}^{16} [f_t(x, y) - f_{t-1}(x, y)]^2 \quad (4.1)$$

VAR 越小相关性越大。当  $\text{VAR} \leq 64$  时，采用帧间模式； $\text{VAR} > 64$  时，一般可采用帧内模式。但如果帧内运动小，而过去帧方差大（部分像素值变化大），也会导致残差较大时，可以采用帧间编码处理，即当  $\text{VAR} > 64$ ，但  $\sigma_{t-1}^2 \geq \text{VAR}$  时采用帧间编码。

其中，

$$\sigma_{t-1}^2 = \frac{1}{256} \sum_{x=1}^{16} \sum_{y=1}^{16} \left\{ f_{t-1}(x, y) - \left[ \frac{1}{256} \sum_{x=1}^{16} \sum_{y=1}^{16} f_{t-1}(x, y) \right] \right\}^2 \quad (4.2)$$

在 H.261 中对此并未作出规定。

##### (2) 帧间编码与运动估计

如按上述判决规则采用帧间编码，以宏块为基础的运动补偿预测编码则被引入。当过去

帧与当前帧的宏块匹配时，求出该亮度宏块的运动矢量  $V_x$  和  $V_y$ ，分别表示过去帧的最佳匹配宏块比当前帧的宏块右移  $V_x$  个像素及下移  $V_y$  个像素，构成当前帧的亮度预测帧；色度块则作  $V_x/2$  和  $V_y/2$  位移，组成当前色度预测帧。接着将相应的当前帧块和预测帧块相减，的帧误差信号（残差信号），再对此帧误差信号进行 DCT 变换、量化后传送到复合编码器。

量化后信号经过反量化的残差图像，如不计量化失真，与预测图像相加得到恢复得原始图像，存入帧存储器。每个宏块的原始图像都存入帧存储器，直到一帧中所有宏块残差都发完。于是当前帧存满  $f_i(x,y)$ ，并作为下一帧  $f_{i+1}(x,y)$  的参考图像。

(3) 帧内编码

如上述判决准则，采用帧内编码，帧内/帧间开关向上。 $f_i(x,y)$  直接经 DCT 变换，变换系数经过量化送入视频复合编码器。

(4) 环路滤波器

由于利用运动补偿方法得出预测图像，宏块匹配可能不准确，导致宏块的运动矢量不准确及方块效应出现，最终产生伪轮廓等高频成分，大大影响了图像质量。为此需设置环路滤波器保证图像质量。**环路滤波器实质上是一个 FIR 低通滤波器，把一些不必要的高频滤去。**

4.2.2.3 视频复合编码器

针对图 4.1 中四层的视频数据结构，视频复合编码器对量化后的视频数据进行变长编码（直流系数用定长编码），并插入一些辅助数据（如帧首、块组首、宏块首等），得到复合编码图像数据结构，如图 4.4 所示。

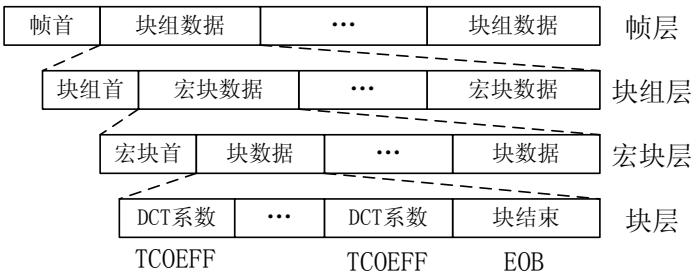


图 4.4 图像复合编码数据结构

各层数据简要说明如下：

- (1) 帧层，帧首包括帧起始码（PSC）、帧计数码（TR）、帧类型信息码（PTYPE）及备用插入信息码等；
- (2) 块组层，块组首包括块组起始码（GBSC）、块组编号码（GN）、块组量化步长（GQOANT）、备用插入信息等；
- (3) 宏块层，由 6 个块组成，包括 4 个亮度块和两个色度块。宏块首包括宏块地址码（MBA）、宏块类型码（MTYPE）、宏块量化步长码（MQUANT）、运动矢量数据（MVD）、编码模式（CBP）等，CBP 说明各块中数据数；
- (4) 块层，各块（8×8）的 DCT 系数按图 4.5 所示的 Zig-Zag（之字形）扫描，将像素排成一维序列并进行熵编码（变长编码），每块结束时用 EoB 结束符作为块结束。

1	2	6	7	15	16	28	29
3	5	8	14	17	27	30	43
4	9	13	18	26	31	42	44
10	12	19	25	32	41	45	54
11	20	24	33	40	46	53	55
21	23	34	39	47	52	56	61
22	35	38	48	51	57	60	62
36	37	49	50	58	59	63	64

图 4.5 之字形扫描

#### 4.2.2.4 传输缓冲器

为使输出码率基本恒定，可采用缓冲器控制并调整帧亚速率、改变量化步长等等。

在 H.261 中，建议缓存器容量 B 应满足下式：

$$B \geq 4R/f_F + 256 \quad (\text{Kb}) \quad (4.3)$$

其中 R 为信道速率， $f_F$  为帧频。

例：在会议电视中，如  $R=2.048\text{Mbps}$ ， $f_F=30$  帧/秒，则  $B \geq (4 \times 2048/30 + 256) \text{Kb} = 529 \text{Kb} = 66.1 \text{KB}$ ，实际取 96KB 为缓存容量。

#### 4.2.2.5 传输编码器

H.261 中规定使用 BCH 码纠正信道中的误码，其码长  $n=511$  位，其中信息码 493 位，即采用 BCH (511, 493) 纠错码。

由于：

$$n=2^m-1, n-k \leq mt \quad (4.4)$$

将  $n=511$ ， $R=493$  代入，得  $m=9$ ， $t=2$ ，即可以纠正 2 位误码。

最后，图 4.6 给出纠错数据帧结构，每个数据帧由  $R=483$  位信息码，再加 1 位同步码构成。一个数据帧共有 512 位。8 个数据帧构成一个帧群，帧首中同步码为 1 位，规定  $S_1S_2S_3 \dots S_8=00011011$ 。

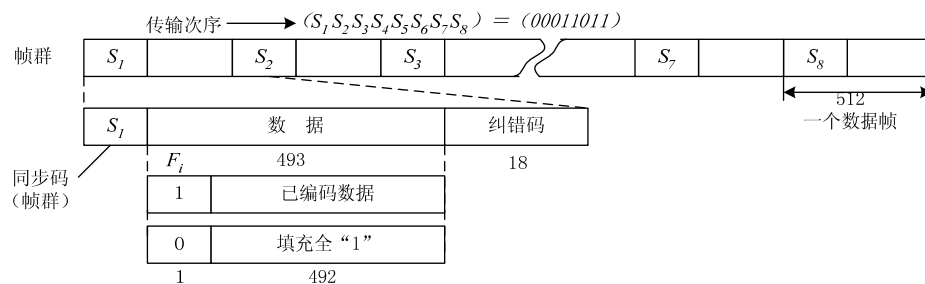


图 4.6 纠错数据帧结构

解码器结构如图 4.7 所示，其原理与编码器相反，这里不再重述。

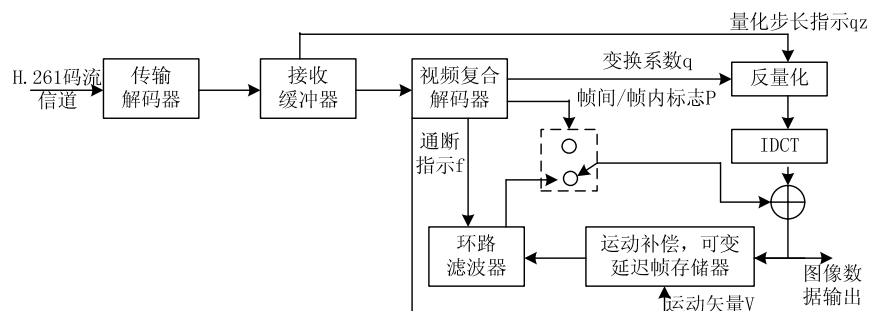


图 4.7 解码器框图

### 4.3 H.263 标准

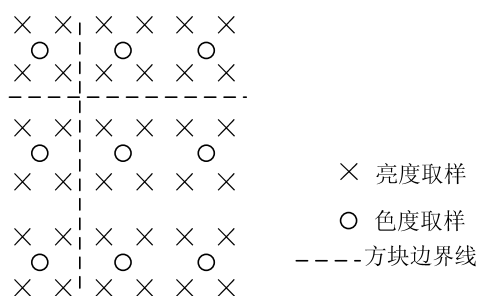
### 4.3.1 H.263 标准图像格式

如表 4.2 所示, H.263 共有五种图像格式。

表 4.2 H.263 标准图像格式

图像格式	亮度信号分量 (Y)		色度信号分量 (Cb 或 Cr)	
	每行像素数	每帧行数	每行像素数	每帧行数
Sub-QCIF	128	96	64	48
QCIF	176	144	88	72
CIF	352	288	176	144
4CIF	704	576	352	288
16CIF	1408	1152	704	576

彩色格式取 4:2:0，亮度、色度取样位置如图 4.8 所示。



1
2
3
4
5
6
7
8

图 4.9 QCIF 图像 GOB 编号

对于 Sub-QCIF、QCIF、CIF 格式，每个块组（GOB）均为 16 行。4CIF 每一块组 32 行，16 CIF 每一块组 64 行。Sub-QCIF、QCIF、CIF、4CIF、16 CIF 含有得块组数分别为 6、9、18、18、18。

QCIF 图像得 GOB 如图 4.9 所示。每个宏块由 4 个  $8 \times 8$  亮度块和 2 个  $8 \times 8$  色度块组成。对于 Sub-QCIF、QCIF、CIF 格式，每个 GOB 垂直方向有一个宏块行，水平方向分别有 8、11、22 个宏块；对于 4CIF 和 16 CIF 垂直方向分别有 2 个及 4 个宏块行，水平方向分别有 44 个和 88 个宏块。

### 4.3.2 H.263 视频信源编码算法

H.263 的视频信源编码框图与 H.261 相同，信源编码方法也类似，不同的是 H.263 输入有多种格式，输出为 H.263 码流。传输码率最初定为低于 64Kbps，但实际上其应用范围已远远超出低码率图像编码范围，如 16QCIF 已是高清晰度电视水平。可以说，H.263 也适于高速率图像编码。

为了适应低码率传输要求，并进一步提高图像质量，H.263+、H.263++做了不少改进，增加了若干选项，现选择主要技术介绍如下：

(1) 运动矢量，H.263 中 1 个 MB 可以使用 1 个运动矢量表示，也可以 4 个 8×8 块各使用 1 个运动矢量表示，提高运动估计精确性和压缩比。(H.261 规定每个 MB 使用 1 个运动矢量。)

(2) 半像素预测，H.263 为进一步提高压缩比，采用半像素预测，而 H.261 采用整像素预测，预测精度明显低于 H.263。

(3) 二维预测，H.263 采用二维预测，H.261 采用一维预测。

(4) 非限制的运动矢量模式（选项），H.263 的运动矢量范围允许指向图像帧之外。

(5) 基于句法的算术编码（选项），显著降低码率，但复杂度比哈夫曼编码高。

(6) 高级预测模式（选项），H.263 除可以采用每个 8×8 块 1 个运动矢量，每个 16×16 宏块 4 个运动矢量外，还采用 OBMC 运动补偿方式，以减少方块效应。

(7) PB 帧模式（选项），PB 帧由 1 个 P 帧和 1 个 B 帧组成。P 帧由前一帧预测而得。B 帧由双向（前向和后向）预测而得，分别用前向 MV、后向 MV、前后向 MV 平均进行运动补偿得 3 个预测误差，取其最小者作为 B 帧的预测误差进行编码。

## 4.4 MPEG-1 标准

### 4.4.1 功能

(1) 视频压缩编码，压缩后码率在 1.5Mbps，可用于视频传输和视频存储；编码前必须将视频图像转换成逐行扫描图像。

(2) 录像机的正放、图像冻结快进、快退和慢放功能以及随机存储功能。

### 4.4.2 图像类型和编码结构

#### 4.4.2.1 MPEG-1 的图像类型

MPEG-1 定义了三种图像类型：I、P、B 图像。I 图像即帧内（Intra）图像，采用帧内编码，不参考其它图像，但可作为其它类型图像的参考帧。P 图像即预测（Predicted）图像，采用帧间编码，参考前一幅 I 或 P 图像，用作运动补偿。B 图像即双向预测（Bi-predicted）图像，参考前后两个方向图像。I、P、B 图像之间的显示顺序见图 4.10。

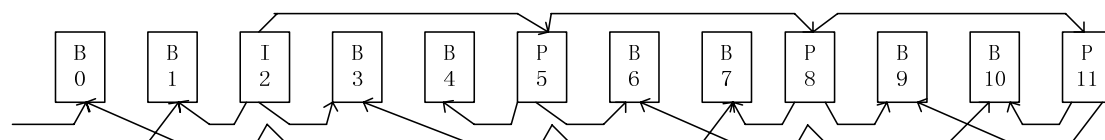


图 4.10 I、P、B 图像显示顺序

双向预测编码可解决“暴露”问题，即某物体在前一帧未显示出来，但在后一帧却“暴露”出来，双向预测能更准确地找出运动矢量，并只有在视频存储、VOD 等非实时通信及数字广播电视中应用。会议电视、可视电话等实时通信中不宜应用 B 图像，因为实时通信后一帧处在当前帧之后，当前帧编码时它尚未出现。

4.4.2.2 MPEG-1 编码结构

MPEG-1 编码结构类似 H.261，也采用分层结构，但有所不同，如图 4.11 所示。可见 MPEG-1 多出片层（切片或 slice 层），用于防止误码在一帧内地扩散。

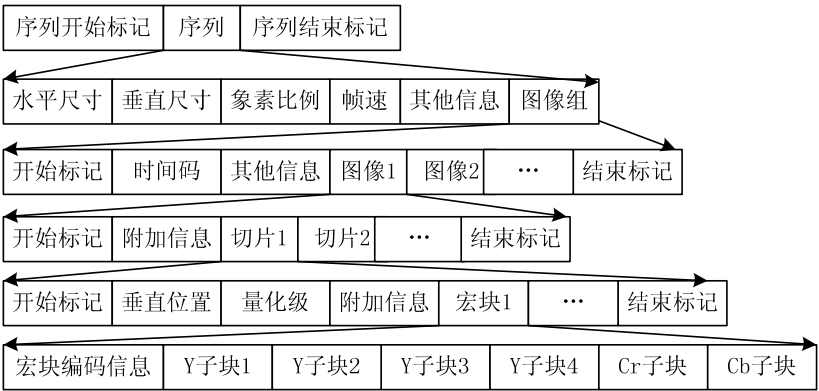


图 4.11 MPEG-1 编码层次结构

4.4.2.3 视频编码器

和 H.261 并无多大区别，但需考虑双向预测编码和解码。解码时先将恢复图像存在缓存中，以显示顺序显示。

4.5 MPEG-2

MPEG-2 标准包括系统、视频、音频及符合性（检验和测试视音频及系统码流）4 个文件。

4.5.1 MPEG-2 编码复用系统

MPEG-2 码流分为三层，即基本流（ES, elementary bit stream）、包基本码流（PES, packet elementary stream）和复用后的传送码流（TS, transport stream）、节目码流（PS, program stream）。其编码复用系统结构如图 4.12 所示。

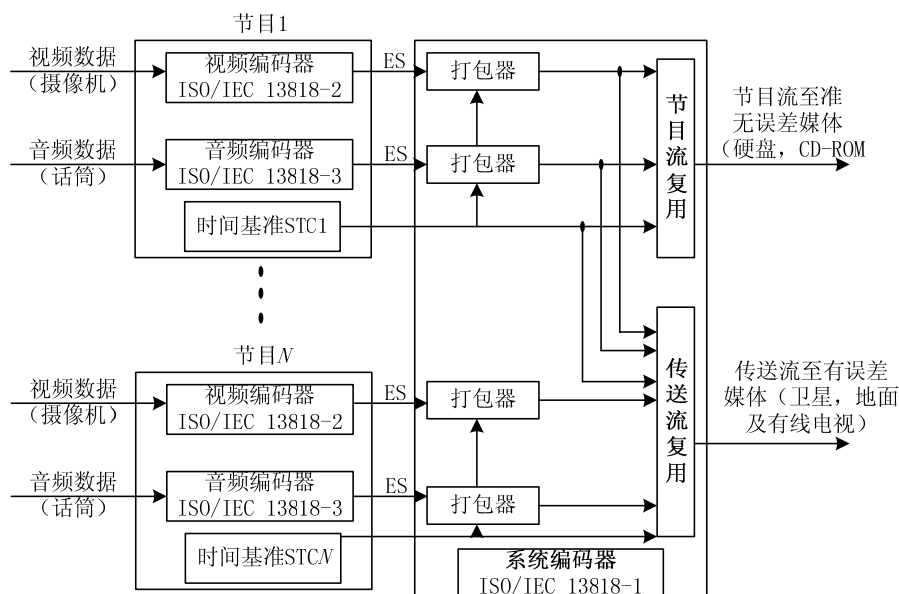


图 4.12 MPEG-2 编码复用系统

ES: 由视频压缩编码后的视频基本码流 (Video ES) 和音频压缩编码后的音频基本码流 (Audio ES) 组成。

PES: 把视音频 ES 分别打包, 长度可变, 最长为  $2^{16}$  字节。

TS、PS: 若干个节目的 PES 复用后输出为传输流 TS 和节目流 PS, 分别用于传输和存储。

## 4.5.2 档次和级别

MPEG-2 按不同的压缩比分成五个档次, 按视频清晰度分为四个级别, 如表 4.3 所示。共有 20 种组合, 最常用的为其中的 11 种组合, 分别用于标准数字电视、高清晰度电视, 码率从 4Mbps 到 100Mbps。

表 4.3 MPEG-2 档次和级别

档次 (profile) 级别 (level)	简单 (simple) SP	主要 (main) MP	SNR 可分级 (scalable) SNP	空间可分级 (spatial scalable) SSP	高级 (high) HP
高级 (high) HL 1920×1080×30 或 1920×1152×25		MP@HL 80Mbps			HP@HL 100Mbps +较低层
高-1440 (high-1440) 1440×1080×30 或 1440×1152×25		MP@H1440 60Mbps		SSP @H1440 60Mbps +较低层	HP @H1440 80Mbps +较低层
主级 (main) ML 720×480×30 或 720×576×25	SP@ML 15Mbps	MP@ML 15Mbps	SNP@ML 15Mbps +较低层		HP@ML 20Mbps +较低层
低级 (low) LL 352×288×29.97		MP@LL 4Mbps	SNP@LL 4Mbps		

### 4.5.3 MPEG-2 视频编码器

#### 4.5.3.1 框图

MPEG-2 编解码器结构与 H.261 区别不大, 如图 4.13 和图 4.14 所示。

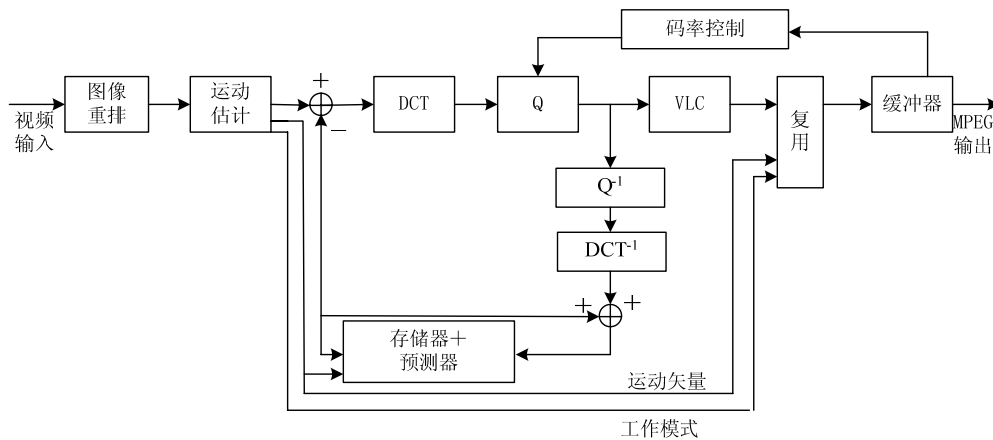


图 4.13 MPEG-2 编码器

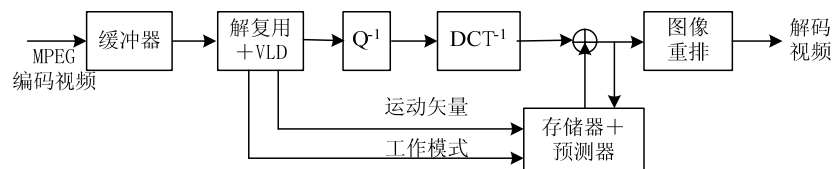


图 4.14 MPEG-2 解码器

#### 4.5.3.2 ES 码流结构

ES 码流采用图像序列 (PS)、图像组 (GOP)、图像 (P)、片 (slice)、宏块 (MB)、块 (B) 六层结构, 具体结构见图 4.15。

- (1) 图像序列层, 图像序列包括若干 GOP, 序列头包含起始码和序列参数, 如档次、级别、彩色图像格式、帧场选择等等;
- (2) 图像组层, 图像组包含若干幅图像, 组头包含起始码、GOP 标志等, 如视频磁带记录器时间、控制码、B 帧处理码等;
- (3) 图像层, 一幅图像包含若干片, 头信息中有起始码、P 标志, 如时间、参考帧号、图像类型、MV、分级等;
- (4) 片层, 片是最小的同步单位, 包含若干宏块, 片头中有起始码、片地址、量化步长等;
- (5) 宏块层, 宏块由 4 个  $8 \times 8$  亮度块和 2 个色度块组成, 宏块头包括宏块地址、宏块类型、运动矢量等。

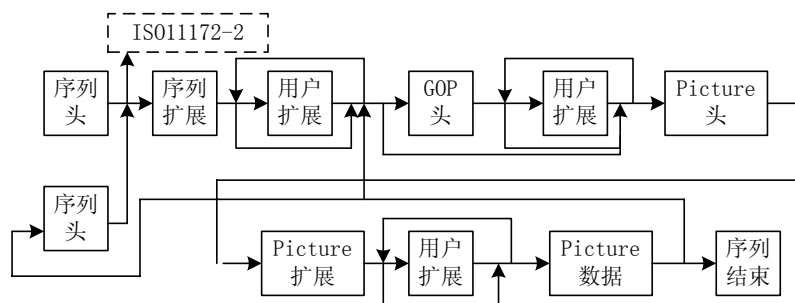


图 4.15 ES 码流结构



### 4.5.3.3 基于帧或场的 DCT 编码

MPEG-2 可用于逐行扫描图像也可用于隔行扫描图像。对逐行扫描图像，可按行分割成块，基于块进行 DCT 变换。对逐行扫描图像，一帧由两场组成，于是就有基于帧的分割和基于场的分割两种宏块结构，如图 4.16 所示。

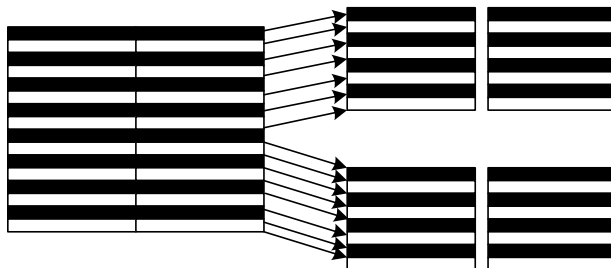


图 4.16 按帧分割进行 DCT 变换的亮度宏块结构

同一帧内的各邻近行之间空间相关性比同一场内各邻近行之间的相关性要强。因此基于帧的 DCT 编码适用于相对静止或慢运动的景物。

同一场内各邻近行之间时间相关性比同一帧内各邻近行之间的时间相关性要强(因为同一帧内当前行的下一行要等到一场扫描完后才出现在当前行之下，时间相关性弱)，基于场的 DCT 编码适用于运动大的景物。

根据帧的行间相关系数和场的行间相关系数可判定采用帧分割还是场分割进行编码。

### 4.5.3.4 分级服务

为了适应信道的变化和扩大应用范围，MPEG-2 采用三种分级编码：空间域分级、时间域分级和信噪比分级。

以空间域分级为例，用户接收 HDTV 信号时，丢弃一部分解码其中的 SDTV 信号，也可再丢弃一部分解出 CIF 甚至 QCIF 信号。这种分级可为不同用户提供不同质量的服务。

## 4.6 JPEG 标准

1991 年 3 月 ISO/IEC 正式通过了静止图像压缩编码标准，称为 JPEG 建议。JPEG 标准分为基本系统、扩展系统和信息保持系统三个部分。

基本系统提供对顺序扫描静止图像的高效有损编码，输入图像精度为 8bit/像素。

(1) 图 4.17 为 JPEG 基本系统的编解码器方框图，输入的彩色图像为 Y、U、V 三个分量，JPEG 对他们分别进行编码。

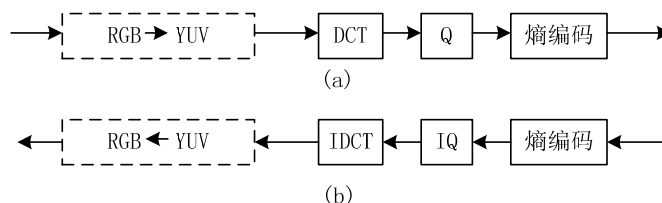


图 4.17 JPEG 基本系统编解码器结构

(2) 编码时，先将一帧图像分为互不重叠的  $8 \times 8$  像素块，接着对各块进行 DCT 变换，然后对各变换系数进行线性量化。

量化步长  $Q$  应结合人眼视觉敏感性，亮度和色差信号的量化步长矩阵见表 4.4 和表 4.5。

表 4.4 亮度量化步长矩阵

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

表 4.5 色差量化步长矩阵

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

量化后系数为：

$$F^*(u,v) = \text{取整} \left[ \frac{F(u,v)}{Q(u,v)} \right] \quad (4.5)$$

反量化后 DCT 系数为：

$$F'(u,v) = F^*(u,v) \times Q(u,v) \quad (4.6)$$

(3) 熵编码，一般采用哈夫曼（VLC）编码。AC 系数量化后为少数稀疏的值，大部分为零，采用锯齿形（Zig-Zag）扫描，然后以游程编码表示方式进行变长的哈夫曼编码。对于 DC 系数则采用本块与上一块 DC 系数之差进行 VLC 编码（可查表）。

(4) 数据交换格式

熵编码后得到变长度的码流。为便于数据的交换，JPEG 规定了统一的压缩后数据交换格式，如图 4.18 所示。

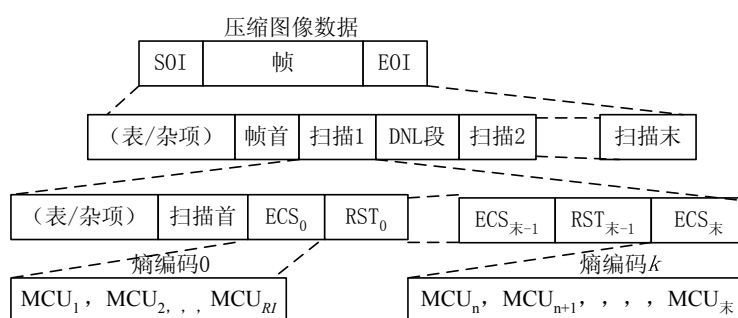


图 4.18 数据交换格式

第一行：SOI 表示图像数据开始；EOI 为一帧图像结束，各占两个字节。

第二行：表/杂项中放置量化表、哈夫曼表；帧首包括编码方法、取样精度、量化系数、源图像行数、每行取样数等；DNL 重新定义帧内的行数。

第三行：扫描首说明扫描起始信息、分量图像号码、参数、熵编码表选择（ECS）；RST 为重新开始标志。

第四行：为熵编码区，MCV 为最小编码单元，包括 4 个亮度块，1 个 Cr 块和 1 个 Cb 块。

## 参考文献

- 1 许志祥, 数字电视与图像通信, 上海大学出版社, 2000
- 2 余兆明, 数字电视和高清晰度电视, 人民邮电出版社, 1997.5
- 3 朱秀昌, 图像通信应用系统, 北京邮电大学出版社, 2003.6
- 4 毕厚杰, 图像通信工程, 邮电出版社, 1995.10

## 第5章 MPEG-4 压缩编码标准

多媒体技术获得使用的关键技术之一就是解决视频、音频数字化后数据量大与数字存储媒体和通信网带宽小的矛盾，为此国际标准化组织先后于 1993 年和 1995 年制定了视音频压缩编码标准 MPEG-1 和 MPEG-2，使数字视频和数字电视成为现实。

MPEG-4 于 1999 年初正式成为国际标准，与前面两个标准相比，MPEG-4 更加注重多媒体系统的交互性和灵活性，主要应用于可视电话、视频会议等。MPEG-4 标准主要包含音视频对象编码工具集和编码对象句法语言两个部分。接受者可以下载用于表示音视频信息的语法描述，并被 VLSI 技术支持。

### 5.1 MPEG4 标准概述

#### 5.1.1 MPEG-4 标准特性

MPEG-4 初衷是针对视频会议、可视电话的超低比特率编码的。而在其调查过程中，高能通用芯片性价比提高使得基于软件平台的压缩编码具有实用可能，且人们在对视频信息的应用需求从播放型逐渐转到基于内容的访问和操作型。为此，MPEG-4 制定新的目标：支持多种多媒体应用（侧重于对多媒体信息内容的访问），可根据应用要求配置解码器。

MPEG-4 标准的编码基于对象，便于操作和控制对象。在比特率控制时，即使在低带宽条件下，MPEG-4 也可利用码率分配方法，对用户感兴趣的对象多分配比特率，对其它则少分配比特率，保证主观质量。MPEG-4 的对象操作使用户可在终端直接将不同对象进行拼接，得到用户合成图像。

MPEG-4 具有很好的扩展性，可进行时域和空域的扩展。这在 MPEG-2 中也有所体现，但不突出。MPEG-4 可根据带宽和误码率的客观条件，在时域或空域进行扩展。前者指在带宽允许时增加帧率带宽窄时，减少帧率，已达到充分利用带宽；后者指对图像进行采样插值，增加或减少空间分辨率。

MPEG-4 有多种算法，可根据需要选择，例如区域编码有 DCT、SADCT、OWT 等等。

MPEG-4 为了支持高效压缩、基于内容交互和基于内容分级扩展，以基于内容的方式表示视频数据，引入 AVO（Audio/Video Object）的概念实现基于内容的表示。

#### 5.1.2 AVO 及数据结构

AVO 基本单位是原始 AV 对象，可能是一个没有背景的说话的人，也可能是这个人的语音或背景音乐等等。它具有高效编码、高效存储传播及可交互操作的特性。

与 MPEG-1 和 MPEG-2 相比，MPEG-4 特点是其更适于交互 AV 服务和远程监控。可以这样说，MPEG-4 就是围绕 AV 对象的编码、存储、传输和组合而制定的。MPEG-4 对 AV 对象的主要操作如下：

- （1）采用 AV 对象表示音视频或其组合内容；
- （2）组合已有 AV 对象，通过自然混合编码 SNHC 组织；
- （3）可对 AV 对象数据多路合成和同步，以便选择合适网络传输数据；
- （4）允许用户对 AV 对象进行交互操作；
- （5）支持 AV 对象知识产权和保护。

MPEG-4 是第一个使用户可在接收端对画面进行操作和交互访问的编码标准。由于 MPEG-4 基于对 AVO 独立编码，必须同时传送编码对象的组成结构信息体“场景描述”，它不属于 AVO 的特征信息，仅仅表示场景中各 AVO 之间的时空结构关系。该信息是独立的，解码时可选定 AVO 的“场景描述”参数，对图像和声音的有关内容进行编辑和操作，如增删某个对象、改变音调、激活分级编码

信息等等。

这里需说明的是，本书主要讲述视频编解码技术，MPEG-4 的 AVO 也相应变为 VO，即视频对象，以下内容也是针对 VO 而言。在 MPEG-4 校验模型中，VO 主要定义为画面中分割出来的不同物体，并由三类信息描述：运动信息、形状信息、纹理信息。

MPEG-4 视频数据流的逻辑结构如图 5.1 所示。

VOP (Video Object Plane, 视频对象平面) 可看作 VO 在某一时刻的表示，即某一帧；GOV (Group of VOPs, 视频对象平面组) 提供视频流的标记点，标记 VOP 单独解码的时域位置，也即对视频流任意访问的标记；

VOL (Video Object Layer, 视频对象层)，用于扩展 VO 的时域和空域分辨率，包含 VO 的三种属性信息；

VO (Video Object, 视频对象) 如前所述，为场景中的某个物体，有生命期，由时间上连续的许多帧构成；

VS (Video Session, 视频镜头)，一个完整的视频序列由几个 VS 组成。

可见，每个 VS 由一个或多个 VO 构成，每个 VO 可能有一个或多个 VOL 层，如基本层、增强层等，每个层是 VO 的某一分辨率表示。每个层中都有时间连续的 GOV，每个 GOV 又由一系列的 VOP 构成。

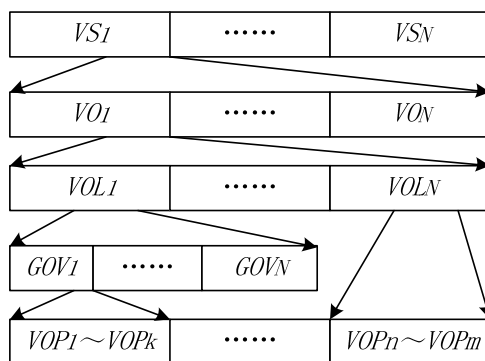


图 5.1 MPEG-4 视频数据流结构

## 5.2 MPEG-4 标准构成

MPEG-4 提供自然和合成的音频、视频以及图形的基于对象的编码工具。类似于以前标准，MPEG-4 由若干部分组成，主要部分为系统、视频和音频。MPEG-4 码流主要包括基本码流和系统流，基本码流包括音视频和场景描述的编码流表示，每个基本码流只包含一种数据类型，并通过各自的解码器解码；系统流则指定根据编码视听信息和相关场景描述信息产生交互方式的方法，并描述其交互通信系统。

### 5.2.1 系统

MPEG-4 系统把音视频对象及其组合复用成一个场景，提供与场景互相作用的工具，使用户具有交互能力。MPEG-4 的系统终端模型如图 5.2 所示。

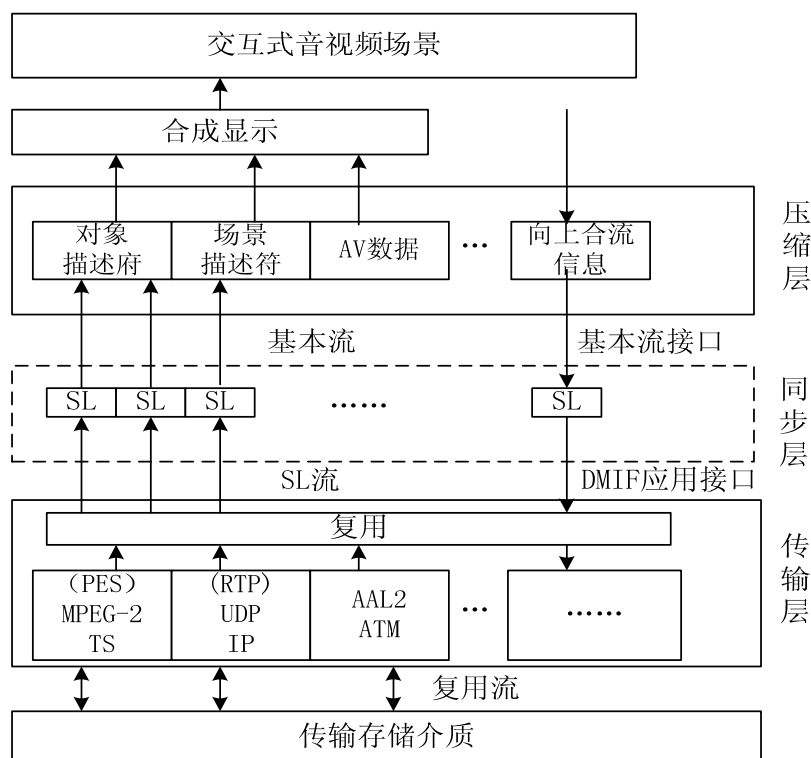


图 5.2 MPEG-4 终端模型

(1) 压缩层，执行媒体解码的系统组件。媒体是通过基本码流接口从同步层提取的。专用 MPEG-4 媒体包括一个二进制格式场景描述符 (BIFS)，用以指定场景合成和图像内容。另一个专用 MPEG-4 媒体类型是对象描述符 (OD)，OD 包含指向基本码流的指针，类似于 URL；OD 也包含附加信息，如服务质量参数等。压缩层不考虑传输的问题。

(2) 同步层，负责各个压缩媒体的同步和缓冲。它接收来自传输层的同步层包 (SL)，根据基本码流的时间标志进行拆包，并转发到压缩层。一个完整的 MPEG-4 节目以不同的基本码流传送每一个媒体类型，如果涉及到可分级性，一些媒体可在几个基本码流中传输。该层通过传送多媒体集成框架 (DMIF) 应用接口 DAI 与传送层对话。

(3) 传输层，对已经存在的各种传输协议描述。这些协议能够用来传输和存储符合 MPEG-4 标准的视听内容。该层的功能并不在系统规定，只是需考虑和传输层有关的接口 DMIF。DMIF 定义了流数据的传输接口及信道建立断开的信号。

系统解码器模型说明符合 MPEG-4 标准的终端功能。发送端可以利用此模型预测接收端在接受到基本码流数据时是如何根据缓冲区管理和同步信息来解码的。系统解码器模型包括定时模型和缓冲模型两种。如图 5.3 所示，每个基本码流都有一个单独的解码缓冲区，单个解码器可以解码多个基本码流（如扩展的视听对象解码）。

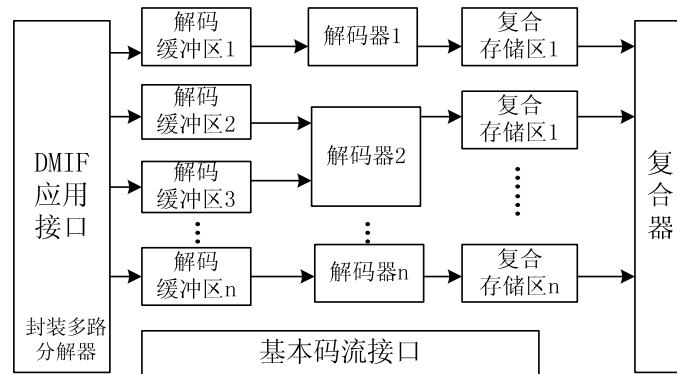


图 5.3 系统解码器模型

## 5.2.2 音频

与 MPEG-1、MPEG-2 相比，MPEG-4 不仅支持自然声音（如语音和音乐），还支持合成声音（如 MIDI）。MPEG-4 音频部分将音频的合成编码和自然声音的编码相结合，并支持音频的对象特征。

### 5.2.2.1 自然声音编码

MPEG-4 研究比较了现有的各种音频编码算法，支持 2Kbps~64 Kbps 的自然声音编码。如 8KHz 采样频率的 2Kbps~4 Kbps 的语音编码，以及 8KHz 或 16KHz 采样频率 4Kbps~16 Kbps 的语音编码，一般采用参数编码；而 6Kbps~24 Kbps 的语音编码，一般采用码激励线性预测 CELP（Code Excited Linear Predictive）编码技术；而从 16Kbps 以上码率的编码，则采用视频变换编码技术。这些技术实质上借鉴了 G.723、G.728 以及 MPEG-1 和 MPEG-2 等。图 5.4 给出了 MPEG-4 音频支持 2~64Kbps 信道语音编码范围。

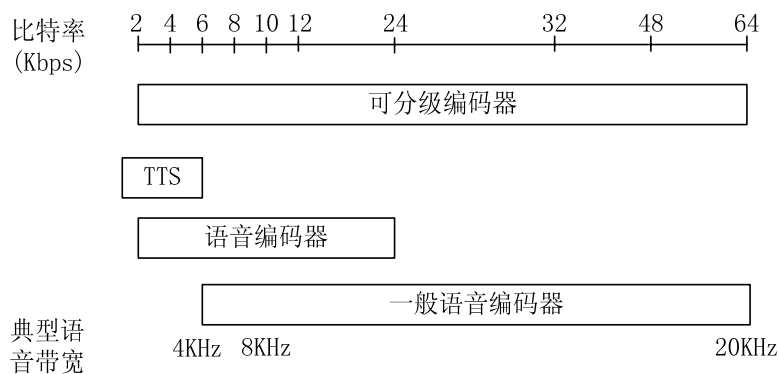


图 5.4 MPEG-4 音频支持语音编码范围

### 5.2.2.2 合成声音编码

MPEG-4 引入两个有力的编码技术：文本到语音编码（TTS，Text-to-Speech）和乐谱驱动合成编码。事实上，合成语音编码技术是一种基于知识库的参数编码。值得一提的是，乐谱驱动合成技术中，解码器由一种特殊的合成语言 SAQL（Structured Audio Orchestra Language，结构化音频管弦乐团语言）驱动的。“管弦乐团”由不同“乐器”组成，解码器不具有某“乐器”时，MPEG-4 还允许解码器从编码器下载该“乐器”，以恢复合成声音。

## 5.2.3 视频

MPEG-4 支持对自然和合成视觉对象的编码。合成的视觉对象包括 2D、3D 动画和人面部表情动画等。对于静止图像，MPEG-4 采用零树小波算法（Zerotree Wavelet Algorithm），以提高压缩比，



同时还提供多达 11 级的空间分辨率和质量的伸缩性。对于运动视频对象的编码，MPEG-4 采用了如图 5.5 所示编码框图，以支持对象的编码。

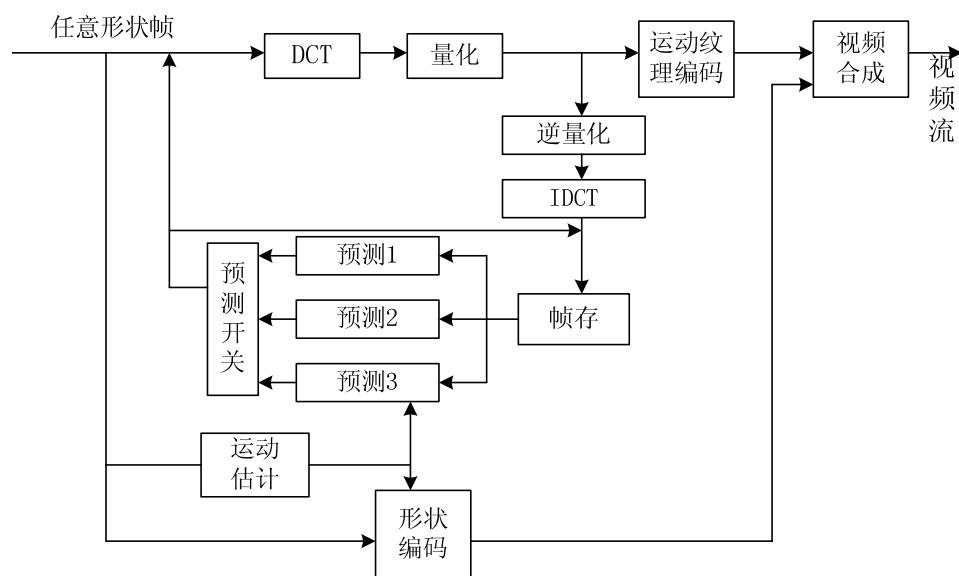


图 5.5 MPEG-4 视频编码框架

### 5.2.3.1 高效的编码工具

MPEG-4 相对 MPEG-1、MPEG-2 而言，编码效率显著提高除了因为基于内容的性质外，还因为引入了以下的编码工具。

- DC 预测，可选择当前块的前一块或者后一块作为当前 DC 值；
- AC 预测，DCT 系数的 AC 预测在 MPEG-4 中是新的。选择用来预测 DC 系数的块也用于预测一行 AC 系数。如果预测器是前一块，则它的第一列的 AC 系数用于预测和当前块相同位置的 AC 系数。如果预测器来自前一块，则用它来预测 AC 系数的第一行。AC 预测对于具有粗糙纹理、对角边缘或水平以及垂直边缘的块效果不佳。在块级切换 AC 预测的通断是所希望的，但这代价太大，一般在宏块级作出。
- 交替水平扫描，这种扫描被添加到 MPEG-2 的两种扫描中。MPEG-2 的交替扫描在 MPEG-4 中被称为交替垂直扫描。交替水平扫描是由镜像垂直扫描得到，在决定 AC 预测的同时选择扫描。在由前一块进行 AC 预测的情况下，选择交替垂直扫描。在由上一块进行 AC 预测情况下，使用交替水平扫描。AC 预测不与“Z”形扫描相结合。
- 三维 VLC，DCT 系数编码与 H.263 类似。
- 四个运动矢量，允许宏块的四个运动矢量，与 H.263 类似。
- 无约束运动矢量，与 H.263 相比，可以使用宽得多的±048 像素的运动矢量范围。
- 子图形，子图形基本上是一个传输到解码器的大背景图像。为了显示，编码器传送该图像的一部分并映射到屏幕上仿射映射参数。通过改变映射，解码器可以放大和缩小子图形，以及向左或向右。
- 全局运动补偿，为了补偿由于摄像机运动、摄像机变焦或者大运动物体引起的全局运动，按照下列公式的八参数运动模型进行补偿：

$$\begin{aligned} x' &= \frac{ax + by + c}{gx + hy + 1} \\ y' &= \frac{dx + ey + f}{gx + hy + 1} \end{aligned} \quad (5.1)$$

全局运动补偿有助于改善最挑剔的场景中的图像质量。

- 四分之一像素运动补偿，主要目的是以小的语法和计算上代价来提高运动补偿的分辨率，得到更精确的运动描述和较小的预测误差。四分之一像素运动补偿只用于亮度像素，色度像素则是用半像素精度运动补偿。

除了为提高编码效率所开发的工具外，MPEG-4 还定义了一系列工具来增强压缩比特流对传输误差的复原能力。这里不作多述。

### 5.2.3.2 基于 VOP 的编码

如前所述，某一时刻 VO 以 VOP 的形式出现，编码也主要针对这个时刻 VO 的形状、运动、纹理这三类信息进行。

- 形状编码

相对以前标准而言，MPEG-4 第一次引入形状编码的压缩算法。编码的形状信息有两种：二值形状信息（Binary Shape Information）和灰度级形状信息（Grey Scale Shape Information）。二值形状信息为用 0、1 的方式表示编码 VOP 形状，0 表示非 VOP 区域，1 表示 VOP 区域；灰度级形状信息可取值 0~255，0 表示非 VOP 区域（即透明区域），1~255 表示透明度不同的区域，255 表示完全不透明。灰度级形状信息的引入主要为了使前景物体叠加到背景上时，边界不至于太明显、生硬，进行“模糊”处理。

MPEG-4 采用位图法表示这两种形状信息。VOP 被一个“边框”框住，如图 5.6 所示，边框长和宽均为 16 的整数倍，同时保证边框最小。位图表示法实际就是一个边框矩阵，取值为 0~255（或 0、1），编码变为对该矩阵的编码。矩阵倍分为 16×16 的形状块，允许进行有损编码，这要通过对边界信息子采样实现，同时允许使用宏块运动矢量作形状块的运动补偿。为了得到语义上更方便的描述，以支持基于内容的操作，MPEG-4 还引入基于上下文的算术编码。

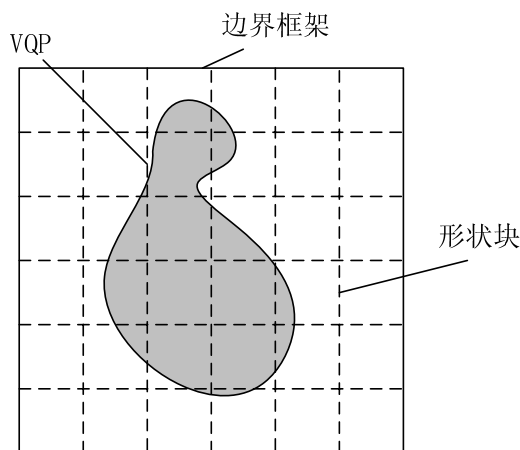


图 5.6 VOP 形状编码

形状编码将在下文详尽阐述。

- 运动估计和运动补偿

类似于以前的压缩标准（MPEG-1、H.263 等）的三种帧格式：I、P、B，MPEG-4 的 VOP 也有三种相应的帧格式：I-VOP、P-VOP、B-VOP（如图 5.7 所示），表示运动补偿类型的不同。如上所述，边框被分为 16×16 的宏块，宏块内是 8×8 的块。运动估计和补偿可以基于宏块，也可基于块。

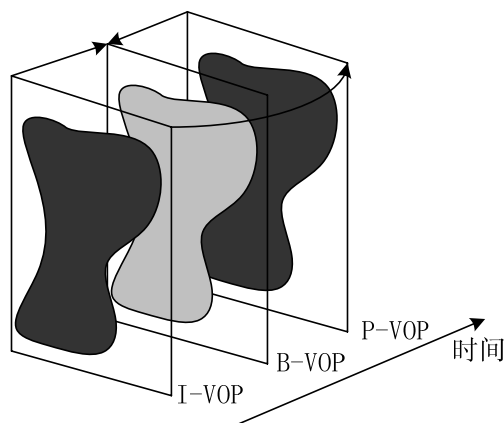


图 5.7 VOP 帧编码类型

MPEG-4 运动估计与补偿类似于 H.263，采用了“重叠运动补偿”。为了使 H.263 的运动估计和补偿算法能适用于任意形状的 VOP 区域，还引入“重复填充”和“修改块（多边形）匹配”技术。

#### ● 纹理编码

纹理信息可能有两种：内部编码的 I-VOP 像素值和帧间编码的 P-VOP、B-VOP 的运动估计残差值。MPEG-4 采用基于分块的纹理编码，VOP 边框仍分为  $16 \times 16$  的宏块。

DCT 变换基于  $8 \times 8$ ，有三种情况：VOP 外和边框内的块不编码；VOP 内的块采用传统 DCT 方法编码；部分在 VOP 内、部分在 VOP 外的块先进行重复填充，再 DCT 编码。这样增加了块内数据的空间相关性，利于 DCT 变换和量化去除块内的空间冗余。

DCT 系数要经量化、Z 扫描、游程及哈夫曼熵编码。量化有两种选择：类似于 H.263 用一个量化参数表征块内所有 AC 系数，这个值可根据质量要求和目标码率变化；或类似于 MPEG-2 用量化矩阵。

#### ● 分级扩展编码

MPEG-4 一个重要功能便是利用 VOL 结构来实现空域和时域的分级扩展。当一个位流中至少存在一个子集能够产生一种有用的表示，则认为这个位流可分级。而视频分级扩展编码就是指可以产生一种编码表示，可支持多于一种的分辨率和质量。

每种类型的分级一般至少都有两个 VOL：一个是基本层，一个是增强层。一般来讲。空域分级就是用增强层来增加基本层的空域分辨率；时域分级就是由增强层来增加基本层中感兴趣区域的时域分辨率，也即帧率。

分级扩展编码将在下文作进一步阐述。

#### 5.2.3.3 VOP 编解码结构框图

如图 5.8 和 5.9 所示，VOP 编解码器主要由两部分组成：形状编解码和传统运动纹理编解码。重构的 VOP 由形状、纹理和运动信息正确组合而成。

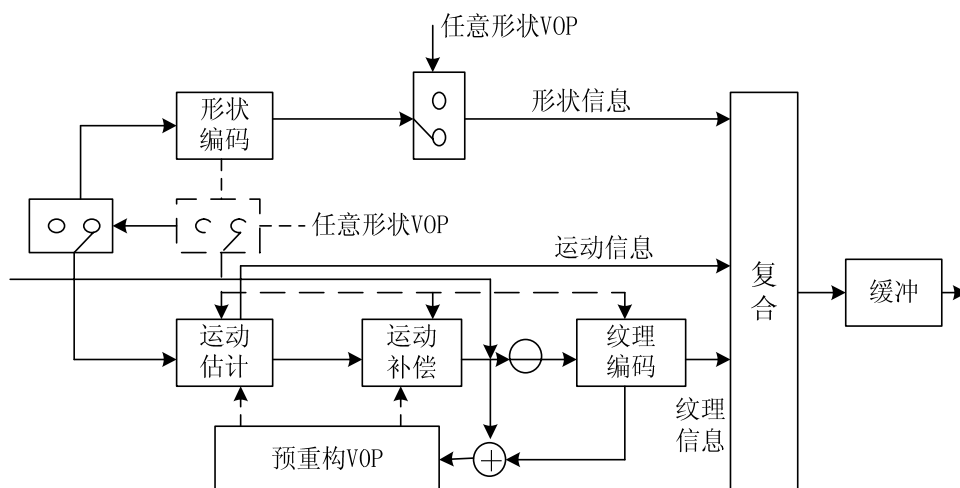


图 5.8 VOP 编码结构

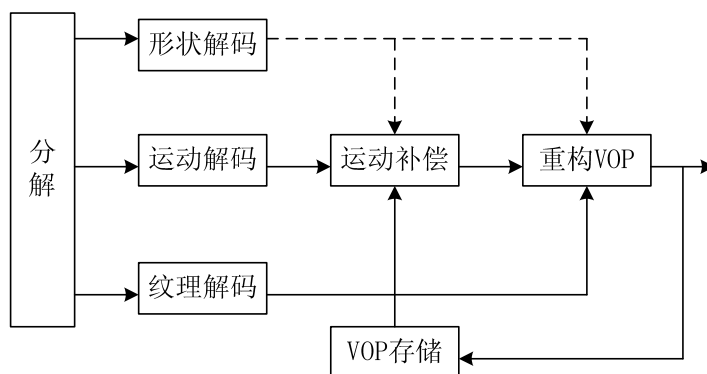


图 5.9 VOP 解码结构

## 5.2.4 网格动画

基于网格的物体表示对许多功能是非常有用的，例如动画、内容操作、内容覆盖、自然与合成视频融合等等。

图 5.10 给出了一个网格编码器以及它与纹理编码器的结合。当一个自然的或合成的视频物体初次出现在场景中时，网格编码器为其产生一个二维基于网格的表示。物体由三角形小块拼接而成，产生一个初始的二维网格。当 VOP 在场景中移动时，这个初始网格的节点就活动起来，或者节点运动由另一个信源激活。视频物体的二维运动可由网格节点的运动矢量紧凑表示。通过对应于各小块的纹理图仿射变换从一个 VOP 变成另一个 VOP 进行扭曲来实现运动补偿。用于映射到物体网格模型或脸部线框模型上的纹理是由视频或静止图像得到的。虽然网格分析不是标准的一部分，但 MPEG-4 定义了二维网格及其节点运动的编码。另外，把纹理映射到网格上可用 MPEG-4 描述。

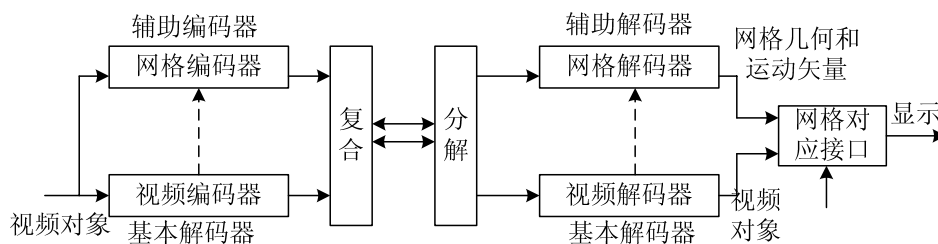


图 5.10 支持二维物体编解码结构

## 5.2.5 其余

- DMIF (Delivery Multimedia Integration Framework)

如前所述, DMIF 即多媒体传送框架, 主要解决交互网络中、广播环境下以及磁盘应用中多媒体应用的操作问题。通过传输多路合成比特信息来建立客户端和服务端端的交互和传输。与以往不同的是, 由于 MPEG-4 码流包括许多 AV 对象, 这些 AV 对象都有各自的缓冲器, 而不仅仅是视频和音频缓冲器。

通过 DMIF, MPEG-4 可以建立具有特殊品质服务的信道和面向每个基本流的带宽。

- 数据平面

MPEG-4 的数据平面可分为两部分: 传输关系部分和媒体关系部分。为了使基本流和 AV 对象在同一场景中出现, MPEG-4 引用了对象描述 (OD) 和流图桌面 (SMT) 的概念。OD 传输与特殊 AV 对象相关的基本流的信息流图。桌面把每一个流与一个 CAT (Channel Association Tag) 相连, CAT 可实现流的顺利传输。

- 缓冲区管理和实时识别

MPEG-4 定义了一个系统解码模式 SDM, 该解码模式描述了一种理想的处理比特流句法语义的解码装置。它要求特殊的缓冲区和实时模式。有效管理可以更好地利用有限的缓冲区空间。

- 场景描述

MPEG-4 提供了一系列工具, 用于组成场景中的一组对象。一些必要的合成信息就组成了场景描述, 这些描述以二进制格式 BIFS (Binary Format for Scene Description) 表示。BIFS 与 AV 对象一同传输、编码。场景描述主要用于描述各 AV 对象在具体 AV 场景坐标下, 如何组织与同步问题。同时还有 AV 对象与 AV 场景的知识产权保护等问题。

## 5.3 MPEG-4 编码技术

### 5.3.1 形状编码

#### 5.3.1.1 MPEG-4 形状编码方法

MPEG-4 的每个 VOP 都包含形状和纹理信息。形状的表现形式主要有二值和灰度两种。二值图 (每像素 8 各比特) 指物体内部点值是 1, 外部点值是 0; 灰度图中的点值可以从 0 到 255 连续变化。MPEG-4 形状编码方法有两种, 如图 5.11 所示。

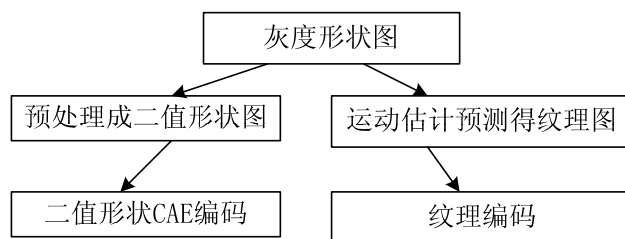


图 5.11 MPEG-4 编码

在解码器端对二值形状 CAE (Content-based Arithmetic Encoding) 解码后, 还需进行泛化 (Feathering) 产生与原来相似的灰度图。

#### 5.3.1.2 二值形状编码方法

二值形状编码基于  $16 \times 16$  的 BAB (Binary Alpha Block) 块的, 其主要步骤如下:

- (1) 对于给定的 VOP 二值形状图重新确定形状边界。

其中，新边界确定原则有：边界框必须是 16×16 的 BAB 块组成；边框左上角的绝对位置坐标为偶数；对 VOP 形状有贡献的 BAB 块数目最小。

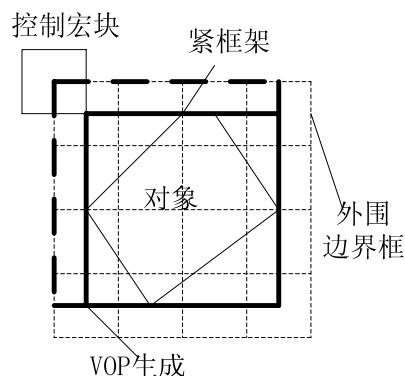


图 5.12 VOP 形状图

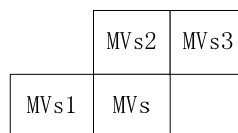


图 5.13 Mvs 预测

(2) 如果该 VOP 是 B-VOP 或 P-VOP，对待编码的 BAB 进行运动估计，得到运动矢量 MVs (MV for shape)；如果该 VOP 是 I-VOP，则该步省略。

MPEG-4 将 MVs 分成两部分，即  $MVs = MVPs + MVDs$ 。其中 MVPs 取当前 BAB 块的左边和上边 BAB 块的 SAD 最小的 MVs；而 MVDs 确定如下：如果 MVPs 所指向的 BAB 块与当前块的残差绝对值之和 SAD 在指定阈值之内，MVDs 为 0，否则在 MVPs 所指定 BAB 块附近搜索得到 MVDs。

(3) 确定该 VOP 中待编码的 BAB 块编码方式。

形状编码共有七种编码方式：1)  $MVDs == 0 \&\& \text{No Update}$ ；2)  $MVDs != 0 \&\& \text{No Update}$ ；3) all\_0；4) all\_255；5) intraCAE；6)  $MVDs == 0 \&\& \text{intraCAE}$ ；7)  $MVDs != 0 \&\& \text{intraCAE}$ 。

对 I-VOP 只有 3)、4)、5) 可以用，对 B/P-VOP 以上七种都可以用。编码方式由码流中 VOP 层的 first\_shape\_code 码字指示。

(4) 确定 BAB 块的分辨率。

由于码率控制，分辨率有时必须改变。尺寸转化有两步。转化比例有 VOP\_CR 确定，VOP 可以取 1 或者 0.5。当 VOP 为 0.5 时，整个 VOP 通过下采样得到原来 1/4 大小的形状图。下采样通过用平均值代替多个采样点，上采样通过插值得到。

(5) 对 BAB 进行编码。

对于 I-VOP，可用 Intra 方式基于上下文的算术编码 (IntraCAE) 编码。对 P-VOP，可用 Inter 方式基于上下文的算术编码 (InterCAE) 编码。基于上下文算术编码基本原理与算术编码相同，主要是将一串相关符号变换到一个数值区间。

### 5.3.1.3 灰度级形状编码

(1) 支持功能和 alpha 值编码

灰度级的 alpha 平面编码由两部分组成：形状轮廓编码和轮廓 alpha 值编码。而轮廓编码采用二值形状编码，alpha 值编码采用任意形状的纹理编码。

轮廓通过在灰度级的 alpha 平面上设定阈值 0 得到。alpha 值被分成 16×16 块并和灰度值类似编码，但 DCT 变换是基于帧的。16×16 块被当作 alpha 宏块，在码流中其编码值附加到对应纹理宏块编码后面。

对于 I-VOP 和 P-VOP，alpha 宏块在码流中位置如下：

CODA	Aacpred_flag	CBPA	Alpha Block
------	--------------	------	-------------

对于 I-VOP 帧内编码的宏块，如果在灰度 alpha 宏块中的所有 alpha 值都是 255（不透明的）和

0，那么 CODA 设为 1。对于 P-VOP 帧间编码宏块，CODA 设定如下，如果 alpha 都为 0，CODA 则为 1，否则，宏块中所有值为 255 时，CODA 为 01，其余则为 00。当 CODA 是 1 或者 01 时，码流中就没有其他 alpha 值得编码。CBPA 表示 alpha 块的编码模式，其他的 alpha 宏块采用纹理宏块编码一样。

对于 B-VOP，alpha 宏块在码流中位置如下：

CODA	MODBA	CBPB	Alpha Block Data
------	-------	------	------------------

这里，如果 alpha 宏块中所有值都为 255，那么 CODA 就设为 1，并且没有其他编码值，否则 CODA 为 0。

## (2) 羽化和半透明编码

许多视频序列使用灰度级的 alpha 掩码，它们纹理相对简单一些，例如由固定灰度级值构成的 alpha 掩码。另外有些掩码在轮廓边缘处有一个从 255 递减到 0 的光滑过渡，这类掩码可以用一个二值的掩码和羽化描述组成。**羽化就是轮廓边缘光滑过渡到背景。**

每个 VOL 描述包含一个选择以下六种模式的识别符：

- ▼ No Effects 模式
- ▼ Linear Feathering 模式
- ▼ Constant Alpha 模式
- ▼ Linear Feathering & Constant Alpha 模式
- ▼ Feathering Filter 模式
- ▼ Feathering Filter & Constant Alpha 模式

对于 Linear Feathering 和 Feathering Filter 模式输入的是一个二值或者灰度级 alpha 掩码序列，Constant Alpha、Linear Feathering & Constant Alpha、Feathering Filter & Constant Alpha 模式则都是灰度级 alpha 掩码序列。如果选择 No Effects 模式，将使用缺省压缩算法。模式用 4 位长度编码，表示符位 video\_object\_layer\_shape\_effects，作为 VOL 层描述符的一部分进行传输，以后还可以扩展其他模式。

对于 Linear Feathering 模式，如果输入的不是二值掩码，那么先将非零值转化为 255，使用一个 0~7 的整数表示将要羽化的范围，该羽化算法将根据这个范围将 alpha 值从 255 线性递减到 0。

对于 Constant Alpha 模式，每个输入灰度级的 alpha 掩码将作为一个固定 alpha 掩码或者标准灰度级掩码来表示。如果掩码中所有非零值都在一个特定范围之内，那么它被认为是一个固定 alpha 掩码。

对于 Feathering Filter 模式，它和 Linear Feathering 模式相似，只是羽化效果是通过滤波实现的，该滤波器是一个非线性的形状自适应滤波器。

## 5.3.2 可扩展性编码

普通意义上讲，视频的可扩展性意味着能够同时得到该视频的多个分辨率或者不同质量，即通过特殊的方法产生一个编码表示，使编码器得到这个视频的多个分辨率或者不同质量的码流。码流的可扩展性则指，解码器可用码流的一部分产生某个分辨率或质量的图像。如果一个码流是可扩展的，那么不同复杂度的解码器可以同时存在，低性能解码器解码产生基本质量的一小部分码流，高性能解码器解码更多码流产生更好质量。

如前所述，扩展主要由两种类型：空域扩展何时域扩展。空域扩展用来扩展空间分辨率，时域扩展则提供时间分辨率的扩展功能。两种类型扩展都涉及多层，一般分为高低两层，低层作为基本层，高层为增强层。传统上，在视频帧上应用这些扩展性，而许多 MPEG-4 要求具有任意形状 VOP 的扩展。

### 5.3.2.1 扩展编码器结构

这里讨论的扩展框架是一种普适性扩展，包含时域和空域扩展。时域扩展时，支持帧和任意形状的 VOP；空域扩展时，只支持矩形 VOP。图 5.14 显示了该扩展的高级编码器结构。

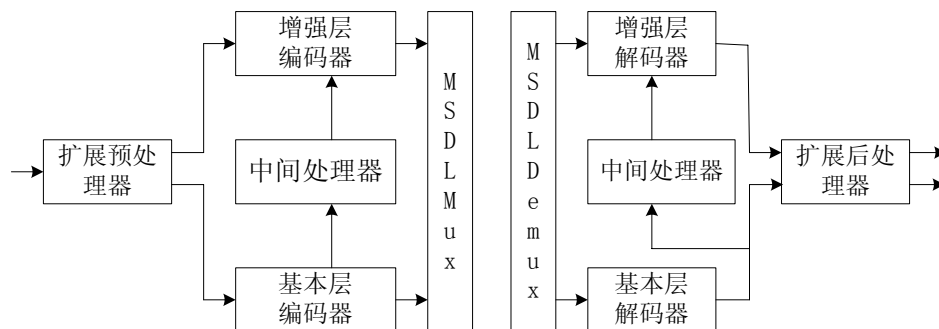


图 5.14 具有扩展性高级编码器结构

图 5.12 中，如果使用空域扩展编码，那么基本层对应着低分辨率，增强层对应着较高分辨率。视频对象 VOP 输入至可扩展预处理器（PreProcessor）中，进行 VOP 空域下采样，产生 MPEG-4 基本层编码器输入，编解码后得到重构 VOP 并输入至中间处理器（MidProcessor1）中，执行空域上采样得到更高分辨率的 VOP 并输入增强层编码器。增强层编码器的另一个输入是预处理器产生的更高分辨率的 VOP，它将参考重构 VOP 进行增强层编码。基本层和增强层码流由 MS D L M u x 混合后存储和传输，基本层和增强层解码器和通过 MS D L D e m u x 访问对应的码流。解码器端的中间处理器（MidProcessor1）和编码器端执行相同操作，扩展编码后处理器（PostProcessor）执行必要的采样转换操作，例如基本层解码后经空域上采样输出等。

当扩展编码使用时域扩展时，可扩展性预处理器在时域上将一个 VO 分解成 VOP 的两个子流，其中一个被输入到基本层编码器中，另一个到增强层中。该情况下，中间处理器不执行任何空域分辨率转换，而是简单地将解码基本层 VOP 输入到增强层编码器中，增强层编码将根据它们进行时域预测。MS D L M u x 和 MS D L D e m u x 操作和空域扩展类似。后处理器也只是简单地输出基本层 VOP，而不进行任何转换，但是在时域上混合基本层和增强层 VOP，以产生更高时域分辨率地增强输出。

### 5.3.2.2 空间域扩展编码

空域扩展时，增强码流用来增加图像分辨率，例如基本层分辨率是 QCIF 大小，而增强层是 CIF 大小。此时若要求输出 QCIF 分辨率，只需要对基本层解码就可以了；如果要求输出 CIF 分辨率，则需要对基本层和增强层同时解码。

如图 5.15 所示，基本层的每帧经上采样，得到增强层的帧，空域分辨率增加，但帧率未增加。对增强层解码前，基本层中对应的参考帧应先解码。

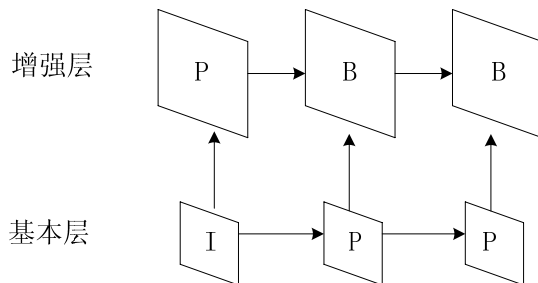


图 5.15 空域分级编码

空域扩展用到技术如下：

- (1) 下采样，由可扩展预处理器执行，采样因子一般取 2。
- (2) 基本层编码，类似于无扩展编码。
- (3) 上采样，由 MidProcessor 完成，重新采样过程是对整个 VOP 定义的，通过参考 VOP 预测



而得。空域预测通过将低层重构的 VOP 重新采样到增强层的分辨率实现。它需要垂直和水平两个方向采样。这在 ISO/IEC 14496-2 规范中定义。

(4) 增强层编码，该层中 VOP 以 P-VOP 或 B-VOP 方式编码。如图 5.15 所示，在时间上与基本层 I-VOP 对应的增强层 VOP 用 P-VOP 方式编码，与基本层 P-VOP 对应的增强层 VOP 用 B-VOP 方式编码。在空域扩展编码时，在基本层解码的 VOP 作为预测的参考 VOP。

空域扩展编码时，不能使用直接编码（类似 H.263 中 B 帧编码）模式。B-VOP 宏块则用前项预测、后向预测及双向预测三种模式编码。

### 5.3.2.3 时间域扩展编码

在基于对象时域扩展编码中，一个被选择的对象帧率可以增强，以取得比其它区域更加平滑的运动效果。该编码有两种类型的增强层结构。图 5.16 显示了一个类型 1 的示意图（使用前项预测的 P-VOP 编码），VOL0 是一个具有对象和背景的完整帧，而 VOL1 表示在 VOL0 的特殊对象，VOL1 的帧率比 VOL0 高。该例中，第 2 帧和第 4 帧是通过结合基本层中的第 0 帧和第 6 帧构成的，这种组合是在组合帧中利用运动预测覆盖所要增强的对象，而该组合帧则是通过背景合成处理形成的。图 5.17 显示了类型 1 的另一个例子，增强层使用 B-VOP 方式编码。需要对两个附加形状信息进行编码，即前向形状和后向形状，用于背景合成。图 5.18 显示了类型 2 的例子，VO0 是一个只包含背景的背景帧，没有扩展层，VO1 是一个特殊对象序列，有两个扩展层 VOL0 和 VOL1，VOL1 用来增强 VOL0 的帧率。

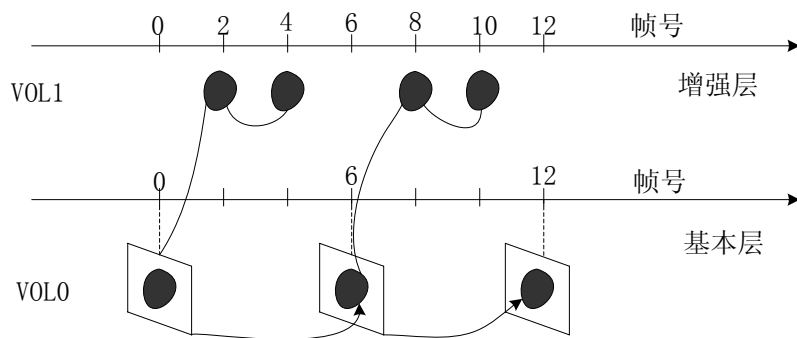


图 5.16 类型 1 的 P-VOP 编码方式示例

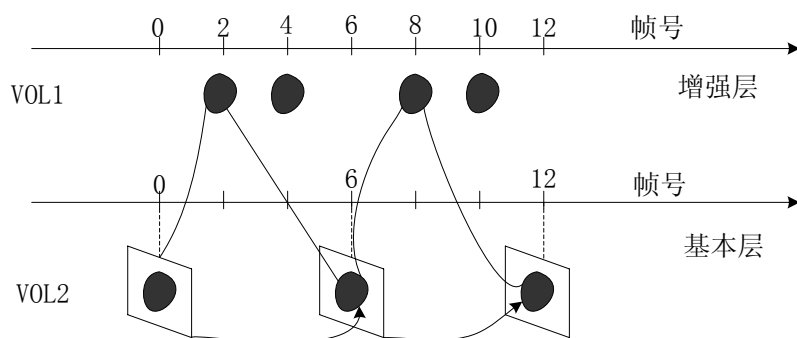


图 5.17 类型 1 的 B-VOP 编码方式示例

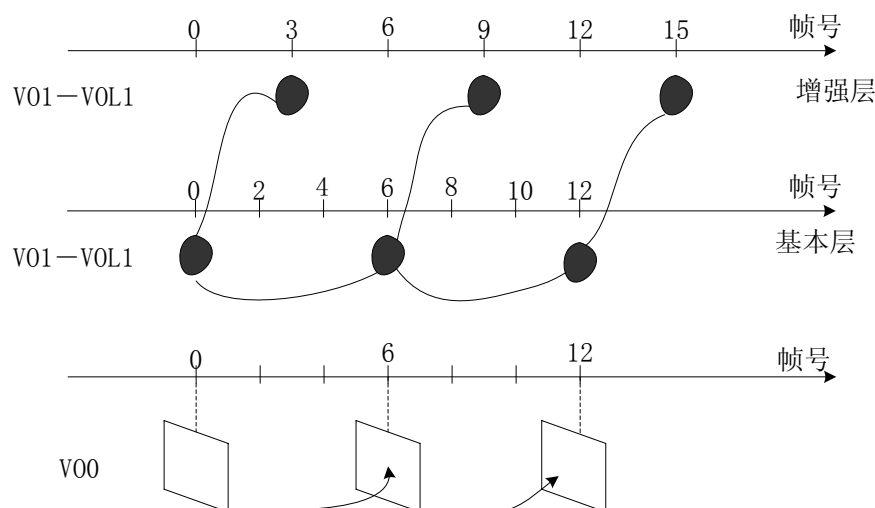


图 5.18 时域扩展类型 2 示例

MPEG-4 可扩展性编码有两种类型的增强：一种是增强层只增强基本层的部分区域质量；另一种是增强层增强基本层的整个区域，由实际应用要求决定。

### 5.3.3 sprite 编码

一个 **sprite** 是由一个视频段中属于同一个视频对象的所有像素构成的，例如从一个全景序列中产生的 **sprite** 将包含整个序列中所有可见的背景对象像素，这个背景中的某些部分在某几帧中可能由于前景对象遮挡或相机运动而使得它们不能被看见。因为 **sprite** 包含了背景中至少出现一次的所有部分，所以它可以用来直接重构背景的视频对象平面 VOP 或者用于背景 VOP 的预测编码。背景 **sprite** 也称为背景全景图。

Sprite 在本质上是一个静态图像，可使用静态图像的传输方法传输，由于 **sprite** 图像一般都比较 大，为了减少传输延迟，可分多次传输，每次只传输一部分，解码器端不断更新已有的 **sprite** 图。

MPEG-4 有两类 **sprite**：静态 **sprite** 和动态 **sprite**。静态 **sprite** 是一种离线图像，基于静态 **sprite** 编码是在某个特定的时刻直接从 **sprite** 图像中复制（包括合适的变形和剪切）对应位置来产生 VOP 显示。而静态 **sprite** 是离线生成的，在视频编码之前按照 I-VOP 的编码方式先编码和传输。离线静态 **sprite** 适合人工合成对象及只有刚体运动自然视频对象。

#### 5.3.3.1 sprite 生成方法

Sprite 生成是 **sprite** 编码的主要部分。Sprite 通常通过全局运动估计生成，除了已经输入一个人工合成对象。Sprite 编码中，色差分量和灰度级的 **alpha** 图和亮度分量相同方式处理，只是参数不同。

另外由于延迟和带宽限制，**sprite** 使用逐块方式传输，先传输最初的低质量图像，然后扩展增强 **sprite** 质量。静态 **sprite** 生成过程如图 5.19 所示。

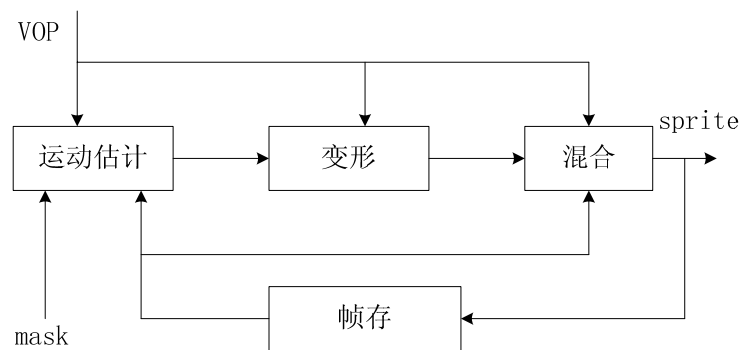


图 5.19 静态 sprite 生成

### 5.3.3.2 sprite 编码

基于静态 sprite 的编码技术是使用指定的运动参数直接将 sprite 变形成得到重构的 VOP, 原始 VOP 和重构的 VOP 之间残差并不编码。在静态 sprite 编码中有一种低延迟编码技术, 主要针对有延迟限制的应用。

减少传输 sprite 延迟有多种办法, 其中一种便是先传输 sprite 中用于重构前几帧的部分, 剩余部分根据解码要求和可用带宽来传输。还有一种办法是先传输一个低分辨率和质量较差的完整 sprite 重构视频序列, 然后在带宽允许条件下传输剩余部分, 以逐步提高 sprite 质量。这两种方法可以单独使用也可以组合使用, 当然在允许情况下可以直接传输 sprite 编码码流, 而不必分层传输。

第一种方法中, sprite 大小、最初块位置、sprite 最初块和整个 sprite 形状信息在 VOL 中传输, 其余部分在 VOP 中传输。在 VOP 中, sprite 剩余部分被分成更小的块, 与轨迹点一同传输。第二种方法逐步提高 sprite 质量。为了提高最初 sprite 质量, 可计算这些区域的残差并作为更新块传输, 在下一个 sprite 块传输延迟允许情况下, 先传输这些更新块。如果带宽允许, 对象和这些更新块可以在同一帧中进行传输。和 sprite 对象传输类似, sprite 残差数据也被分成小块和轨迹数据一起传输, 残差数据编码可以根据质量要求调整。

## 5.3.4 视频系统合成

### 5.3.4.1 时域扩展性合成

时域扩展增强层对象形成背景时会用到背景合成, 即增强层 VOP 与参考 VOP 部分区域相对应时, 将参考层显示顺序中前一个和后一个 VOP 合成作为当前增强层 VOP 的背景。

图 5.20 显示了在增强层中当前帧的背景合成。实线表示参考层的前一个 VOP 选择对象的形状 (前向形状), 虚线表示参考层后一个 VOP 的选择对象形状 (后向形状)。在这些形状之外区域, 合成帧像素值用基本层中最近帧中的对应值; 对象只在前一帧中占据的区域用下一帧中的对应值, 另一方面, 所选对象在下一帧中占据的区域像素值用前一帧中的对应值; 两帧中的对象重叠区域用周围区域的像素值填充。

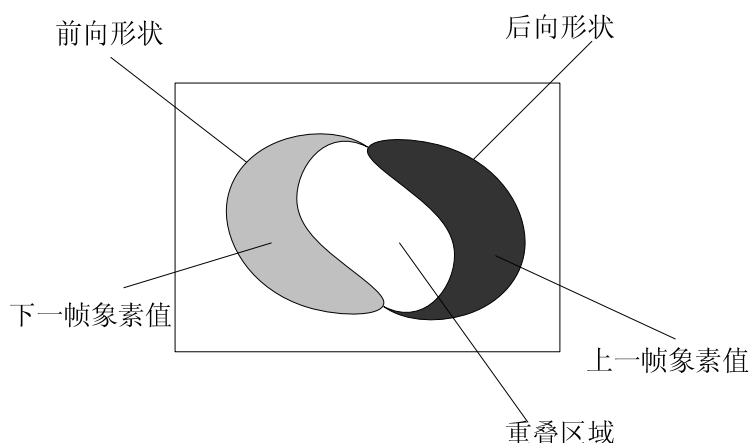


图 5.20 背景合成

背景合成方法如下：

```

If  $s(x,y,t_a)=0$  and  $s(x,y,t_d)=0$ 
     $fc(x,y,t)=f(x,y,t_d)$  ( $|t-t_a|>|t-t_d|$ )
     $fc(x,y,t)=f(x,y,t_a)$  (其余)
If  $s(x,y,t_a)=1$  and  $s(x,y,t_d)=0$ 
     $fc(x,y,t)=f(x,y,t_d)$ 
    If  $s(x,y,t_a)=0$  and  $s(x,y,t_d)=1$ 
         $fc(x,y,t)=f(x,y,t_a)$ 
    If  $s(x,y,t_a)=1$  and  $s(x,y,t_d)=1$ 
         $fc(x,y,t)$ 像素值用周围区域填充。
  
```

其中， $fc$  表示合成图像， $f$  表示基本层解码图像， $s$  表示形状信息， $(x,y)$  表示点空间坐标， $t$  为当前帧时刻， $t_a$  表示前一帧时刻， $t_d$  表示下一帧时刻。 $s(x,y,t_a)$ 称为前向形状， $s(x,y,t_d)$ 称为后向形状。

#### 5.3.4.2 Sprite 合成

如前所述，静态 sprite 技术可有效编码视频对象，它的内容可能不在视频序列的任何时刻出现，适合表示场景背景。

ISO/IEC14496-2 语法定义了从静态 sprite 获得 VOP（称 S-VOP）编码方式，即通过由参数控制处理从静态 sprite 抽取获得。与其它 VOP 合成时，S-VOP 没有空域限制，经常作为背景添加对象。

#### 5.3.4.3 网格对象合成

一个网格对象表示 2D 三角形网格序列的几何形状。这些数据和编码图形纹理通过 ISO/IEC14496-1 定义的合成过程得到纹理映射图像。网格对象流包含在 ISO/IEC14496-1 定义的 BIFS 流中。在实现网格操作的终端中，解码网格数据用来更新 BIFS IndexedFaceSet2D 结点的适当区域。更新过程如下：

- (1) 从网格对象解码器获得网格结点坐标。
- (2) 从网格解码器获得网格结点索引。
- (3) 根据内部编码网格对象平面和范围矩形结点位置，计算纹理映射到网格几何形状纹理坐标。
- (4) 得出纹理坐标索引（同结点坐标索引），并对结点适当区域更新。

## 5.4 MPEG-4 档次和级

这里着重介绍MPEG-4 Visual档次及其相应级。MPEG-4 Visual通过工具、对象和档次的联合提供其编码函数。工具是一些支持一特定特征（如基本视频编码、编码对象形状等）的编码函数集合。对象是利用工具编码的视频元素（如矩形帧、静态图像等）。举个例子来说，一个简单视频对象用针对矩形视频帧序列的工具编码，而一核心视频对象用针对任意形状对象的工具编码等等。档次则是对对象类型的集合，是相应编解码器必须支持的。

MPEG-4 Visual档次中针对自然视频场景编码部分如表5.1所示，其档次范围从针对矩形视频帧编码的Simple档次到了针对任意形状及扩展对象、工作室级视频编码的档次。表5.2列出了人工合成视频和混合视频编码档次。

表 5.1 MPEG-4Visual 自然视频档次

档 次	主 要 特 征
Simple	矩形视频帧低复杂度编码
Advanced Simple	较高效率的矩形帧编码，支持隔行扫描视频
Advanced Real-Time Simple	实时流矩形帧编码
Core	任意形状视频对象的核心编码
Main	多特征视频对象的主要编码
Advanced Coding Efficiency	高效的视频对象编码
N-Bit	N 比特的视频对象编码
Simple Scalable	矩形帧扩展分级编码
Fine Granular Scalability	高级矩形帧扩展分级编码
Core Scalable	视频对象扩展核心可分级编码
Scalable Texture	静态纹理扩展分级编码
Advanced Scalable Texture	高效的、基于对象的静态纹理扩展编码
Advanced Core	包含 Simple、 Core、 Advanced Scalable Texture 档次所有特征
Simple Studio	基于对象的高质量视频序列编码
Core Studio	基于对象的高压缩率高质量核心视频编码

表 5.2 MPEG-4Visual 合成视频档次

档 次	主 要 特 征
Basic Animated Texture	静态纹理二维编码
Simple Face Animation	动态人脸模型
Simple Face and Body Animation	动态人脸和身体模型
Hybrid	上面三种综合

表 5.3 MPEG-4 Visual 档次和视觉对象类型

档 次	对象类型															
	Simple	Advanced Simple	Advanced Real-Time Simple	Core	Main	Advanced Coding Efficiency	N-bit	Simple Scalable	Fine Granular Scalability	Core Scalable	Scalable Texture	Advanced Scalable Texture	Simple Studio	Core Studio	Simple Face Animation	Simple Face and Body Animation
Simple	✓															
Advanced Simple	✓	✓														
Advanced Real-Time Simple	✓		✓													
Core	✓			✓												
Advanced Core	✓			✓								✓				
Main	✓			✓	✓						✓					
Advanced Coding Efficiency	✓			✓		✓										
N-bit	✓			✓			✓									
Simple Scalable	✓							✓								
Fine Granular Scalability	✓	✓							✓							
Core Scalable	✓			✓				✓		✓						
Scalable Texture											✓					
Advanced Scalable Texture												✓				
Simple Studio													✓			
Core Studio													✓	✓		
Basic Animated Texture											✓				✓	
Simple Face Animation															✓	
Simple FBA																✓
Hybrid	✓			✓							✓				✓	✓

表 5.4 Simple 档次级

档次	级	格式	最大比特率	最多对象
Simple	L0	176 × 144	64 kbps	1 simple
	L1	176 × 144	64 kbps	4 simple
	L2	352 × 288	128 kbps	4 simple
	L3	352 × 288	384 kbps	4 simple
Advanced Simple (AS)	L0	176 × 144	128 kbps	1 AS or simple
	L1	176 × 144	128 kbps	4 AS or simple
	L2	352 × 288	384 kbps	4 AS or simple
	L3	352 × 288	768 kbps	4 AS or simple
	L4	352 × 576	3 Mbps	4 AS or simple
	L5	720 × 576	8 Mbps	4 AS or simple
Advanced Real-Time Simple (ARTS)	L1	176 × 144	64 kbps	4 ARTS or simple
	L2	352 × 288	128 kbps	4 ARTS or simple
	L3	352 × 288	384 kbps	4 ARTS or simple
	L4	352 × 288	2 Mbps	16 ARTS or simple

表 5.3 列出了 MPEG-4 Visual 档次和视觉对象类型。该表显示了每个档次包含的对象类型。例如，支持 Simple 档次的编解码器必须能编解码 Simple 对象，而支持 Core 档次的编解码器必须能编解码 Simple 和 Core 对象。

档次是不同制造商编解码器件之间能够交互的重要机制。MPEG-4 Visual 标准定义了多种编码工具，而并非每个商业编解码器件都要支持所有的工具。相反，设计者通常根据应用情况选择包含所需工具的档次。例如，基于低性能处理器的基本编解码器可能用 Simple 档次，而流视频应用编解码器则会选择 ARTS 档次等等。

档次定义了编码工具的集合，级则定义了比特流参数的限制。表 5.4 列出了常用的基于 Simple 档次（Simple、Advanced Simple 及 Advanced Real Time Simple）的级。每个级都给出了能够解码 MPEG-4 编码序列所要求的最大性能。例如，一个只有有限处理能力和内存的多媒体终端只能支持 Simple 档次 0 级的比特流解码。级定义限制了缓冲大小、解码帧大小、处理速度（宏块每秒）以及视频对象数目。终端如能支持这些参数便能解码符合 Simple 档次 0 级的比特流。Simple 档次的更高级则要求解码器支持 4 个 Simple 档次视频对象，如包括 CIF 或者 QCIF 显示分辨率的 4 个矩形对象。

## 参考文献

- 1 Yao Wang, Jorn Ostermann, Ya-Qin Zhang, Video Processing and Communication, Pearson Education, 2002.
- 2 Iain E.G.Richardson, H.264 and MPEG-4 Video Compression Video Coding for Next Generation Multimedia, Wiley Press , 2003
- 3 毕厚杰, 多媒体信息的传输与处理, 人民邮电出版社, 1999
- 4 钟玉琢等.《基于对象的多媒体数据压缩编码国际标准——MPEG-4 及其校验模型》.北京: 科学出版社, 2000



## 第 6 章 H.264/AVC 编码器原理

### 6.1 H.264/AVC 的应用

MPEG (Moving Picture Experts Group) 和 VCEG (Video Coding Experts Group) 已经联合开发了一个比早期研发的 MPEG 和 H.263 性能更好的视频压缩编码标准,这就是被命名为 AVC(Advanced Video Coding)的,也被称为 ITU-T H.264 建议和 MPEG-4 的第 10 部分的标准,在这里,就简称它为 H.264/AVC 或 H.264。这个国际标准已于 2003.3 正式被 ITU-T 所通过并在国际上正式颁布。

应该说, H.264 的颁布是视频压缩编码学科发展中的一件大事, 它的优异的压缩性能也将在数字电视广播、视频实时通信、网络视频流媒体传递以及多媒体短信等各个方面发挥重要作用。

数字电视的优越性已是公认的,但它的广泛应用还有赖于高效的压缩技术。例如利用 MPEG-2 压缩的一路高清晰度电视 (HDTV), 约需 20Mb/s 的带宽, 有人作过初步试验, 如利用 H.264 进行一路 HDTV 的压缩, 大概只需 5Mb/s 的带宽。众所周知, 美国已公布在 2010 年 (我国约在 2015 年) 停止模拟电视广播, 全部采用数字电视广播, 如果那时 HDTV 要获得迅猛发展, 必须要降低成本。以传输费用而言, 采用 H.264, 可使传输费用降为原来的 1/4, 这是一个十分诱人的前景, 据了解, 这次 2008 年在我国北京举行的奥运会, 也将是一个“科技奥运”, HDTV 必然将呈现在人们的眼前一个高质量的压缩性能优异的 H.264 视频编码技术和设备的市场前景是可以想象的! 现在有的省市 (如南京) 已在有线电视信道上开通了数字电视, 采用压缩性能优异的 H.264 显得更为迫切!

视频通信是 H.264 又一个重要应用, 上世纪 90 年代初以来, 会议电视在我国获得了迅速发展, 主要是利用它召开行政会议, 其优点是节约大量旅途出差时间, 节约出差费用, 还争取了时间及时作出了重大决策, 短短几年, 全国从中央到省, 到地市甚至县, 建立了几千个会议电视室, 在国民经济的发挥了重要作用, 其不足之处为: (1) 不方便: 必须到电信局专门的电视会议室才能参加会议, 这对一些领导同志更是十分不便; (2) 价格昂贵: 当时采用 H.261 作为视频压缩编码标准, 压缩比不高, 而且图像质量也不够好, 设备价格昂贵, 传输费用也昂贵; 可视电话是视频通信的另一个重要应用, 人们一直把它作为实现古人的“千里眼、顺风耳”理想的通信工具, 可是直到今天, 尚未很好地广泛地被应用, 其中一个重要原因是视频质量不理想, 这与视频压缩技术有密切关系。特别是, 由于互联网在 90 年代的迅猛发展, 人们希望利用 IP 技术传输视频, 现在人们已可看到, 在网络流量不大时, 人们看到的可视电话质量尚能接受 (尽管也不是很好), 由于 IP 数据流的突发性, 当流量大时, 网络会发生拥塞, 这时经常发生丢包、误码, 看到图像中带有不少方块, 这样的视频质量是无法让人们接受的, 于是对视频编码的要求, 不仅仅要高压缩比, 而且应在恶劣的传输条件下 (包括移动网络的衰落) 具有抗阻塞、抗误码的鲁棒性。

**H.264 不仅具有优异的压缩性能, 而且具有良好的网络亲和性**, 这对实时的视频通信是十分重要的。现在已有基于 DSP 的采用 H.264 编码的可视电话出现在市场上, 进一步说明了在视频通信中 H.264 的重要应用价值。

H.264 还有一个重要应用, 即网络的流媒体。众所周知, 应用流媒体技术的电视点播 (VOD) 最近有了迅速发展, 韩国的宽带上网的应用中 VOD 占据了第二位。我国宽带上网用户今年已达 1000 万户以上, 而且还在继续发展, VOD 的迅速发展也是可以期待的。

多媒体短信息也是 H.264 的重要应用之一, 我国的短信市场正方兴未艾, 相信多媒体短信也将有巨大发展。

和 **MPEG-4 中的重点是灵活性不同, H.264 着重在压缩的高效率和传输的高可靠性**, 因而其应用面十分广泛, 具体说来, H.264 支持三个不同档次:

- 1、基本档次: 主要用于“视频会话”, 如会议电视, 可视电话, 远程医疗、远程教学等;
- 2、扩展档次: 主要用于网络的视频流, 如视频点播;
- 3、主要档次: 主要用于消费电子应用, 如数字电视广播, 数字视频存储等。

## 6.2 H.264/AVC 编解码器

### 6.2.1 H.264 编解码器特点

H.264 并不明确地规定一个编解码器如何实现，而是规定了一个编了码的视频比特流的句法，和该比特流的解码方法，各个厂商的编码器和解码器在此框架下应能够互通，在实现上具有较大灵活性，而且有利于相互竞争。

H.264 编码器和解码器的功能组成分别见图 6.1，6.2。

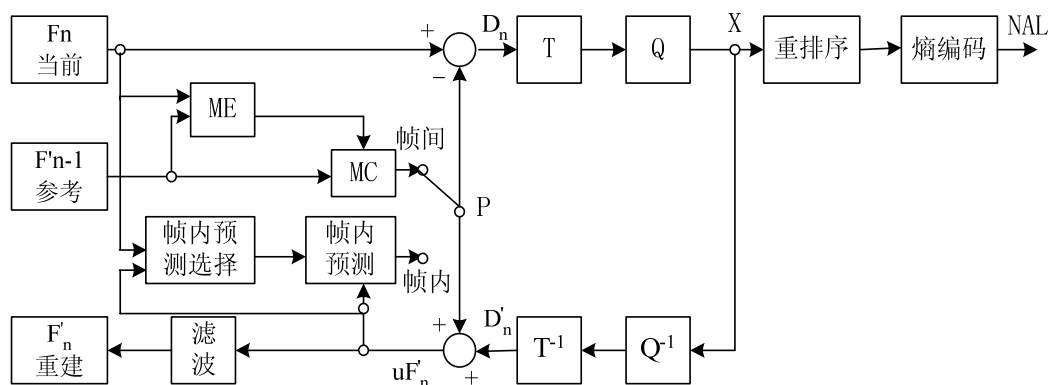


图 6.1 H.264 编码器

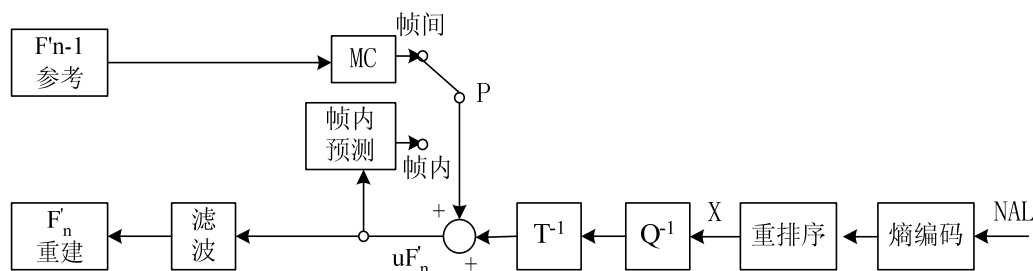


图 6.2 H.264 解码器

从上述二图可见，H.264 和基于以前的标准（如 H.261、H.263、MPEG-1、MPEG-4）中的编解码器功能块的组成并没有什么区别，主要的不同在于各功能块的细节。由于视频内容时刻在变化，有时空间细节很多，有时大面积的平坦。这种内容的多变性就必须采用相应的自适应的技术措施；由于信道在环境恶劣下也是多变的，例如互联网，有时畅通，有时不畅，有时阻塞，又如无线网络，有时发生严重衰落，有时衰耗很小，这就要求采取相应的自适应方法来对抗这种信道畸变带来的不良影响。这两方面的多变带来了自适应压缩技术的复杂性。H.264 就是利用实现的复杂性获得压缩性能的明显改善。由于大规模集成电路技术和工艺的迅猛进步，今天已完全具备了实现的可能性。

在描述多功能块的细节前，我们还是对 H.264 编码器、解码器的主要功能描述如后，以便对编码器有一个总的了解。

### 6.2.2 H.264 编码器

编码器采用的仍是变换和预测的混合编码法。

由图 6.1，输入的帧或场  $F_n$  以宏块为单位被编码器处理。首先，按帧内或帧间预测编码的方法进行处理。

如果采用帧内预测编码，其预测值  $PRED$ （图中用  $P$  表示）是由当前片中前面已编码的参考图像经运动补偿（MC）后得出，其中参考图像用  $F'_{n-1}$  表示。为了提高预测精度，从而提高压缩比，实

际的参考图像可在过去或未来（指显示次序上）已编码解码重建和滤波的帧中进行选择。

预测值 PRED 和当前块相减后，产生一个残差块  $D_n$ ，经块变换、量化后产生一组量化后的变换系数  $X$ ，再经熵编码，与解码所需的一些边信息（如预测模式量化参数、运动矢量等）一起组成一个压缩后的码流，经 NAL（网络自适应层）供传输和存储用。

正如上述，为了提供进一步预测用的参考图像，编码器必须有重建图像的功能。因此必须使残差图像经反量化、反变换后得到的  $D_n'$  与预测值  $P$  相加，得到  $uFn'$ （未经滤波的帧）。为了去除编码解码环路中产生的噪声，为了提高参考帧的图像质量，从而提高压缩图像性能，设置了一个环路滤波器，滤波后的输出  $F_n'$  即重建图像可用作参考图像。

### 6.2.3 H.264 解码器

由图 6.1 可知，由编码器的 NAL 输出一个压缩后的 H.264 压缩比特流。由图 6.2，经熵解码得到量化后的一组变换系数  $X$ ，再经反量化、反变换，得到残差  $D_n'$ 。利用从该比特流中解码出的头信息，解码器就产生一个预测块 PRED，它和编码器中的原始 PRED 是相同的。当该解码器产生的 PRED 与残差  $D_n'$  相加后，就产生  $uFu'$ ，再经滤波后，最后就得到滤波后的  $F_n'$ ，这个  $F_n'$  就是最后的解码输出图像。

## 6.3 H.264/AVC 的结构

### 6.3.1 名词解释

为了弄清 H.264 编解码器的细节，必须先对以下名词的定义有清楚的理解：

#### 1) 场和帧

视频的一场或一帧可用来产生一个编码图像。通常，视频帧可分成两种类型：连续或隔行视频帧。在电视中，为减少大面积闪烁现象，把一帧分成两个隔行的场。显然，这时场内邻行之间的空间相关性较强，而帧内邻近行空间相关性强，因此活动量较小或静止的图像宜采用帧编码方式，对活动量较大的运动图像则宜采用场编码方式。

#### 2) 宏块、片

一个编码图像通常划分成若干宏块组成，一个宏块由一个  $16 \times 16$  亮度像素和附加的一个  $8 \times 8$  Cb 和一个  $8 \times 8$  Cr 彩色像素块组成。每个图像中，若干宏块被排列成片的形式。

**I 片只包含 I 宏块，P 片可包含 P 和 I 宏块，而 B 片可包含 B 和 I 宏块。**

I 宏块利用从当前片中已解码的像素作为参考进行帧内预测（不能取其它片中的已解码像素作为参考进行帧内预测）。

P 宏块利用前面已编码图像作为参考图像进行帧内预测，一个帧内编码的宏块可进一步作宏块的分割：即  $16 \times 16$ 、 $16 \times 8$ 、 $8 \times 16$  或  $8 \times 8$  亮度像素块（以及附带的彩色像素）；如果选了  $8 \times 8$  的子宏块，则可再分成各种子宏块的分割，其尺寸为  $8 \times 8$ 、 $8 \times 4$ 、 $4 \times 8$  或  $4 \times 4$  亮度像素块（以及附带的彩色像素）。

B 宏块则利用双向的参考图像（当前和未来的已编码图像帧）进行帧内预测。

### 6.3.2 档次和级

H.264 规定了三种档次，每个档次支持一组特定的编码功能，并支持一类特定的应用。

1) 基本档次：利用 I 片和 P 片支持帧内和帧间编码，支持利用基于上下文的自适应的变长编码进行的熵编码（CAVLC）。主要用于可视电话、会议电视、无线通信等实时视频通信；

2) 主要档次：支持隔行视频，采用 B 片的帧间编码和采用加权预测的帧内编码；支持利用基于上下文的自适应的算术编码（CABAC）。主要用于数字广播电视与数字视频存储；

3) 扩展档次：支持码流之间有效的切换（SP 和 SI 片）、改进误码性能（数据分割），但不支持隔行视频和 CABAC。

图 6.3 为 H.264 各个档次具有的不同功能，可见扩展档次包括了基本档次的的所有功能，而不能包括主要档次的。每一档次设置不同参数（如取样速率、图像尺寸、编码比特率等），得到编解码器性能不同的级。

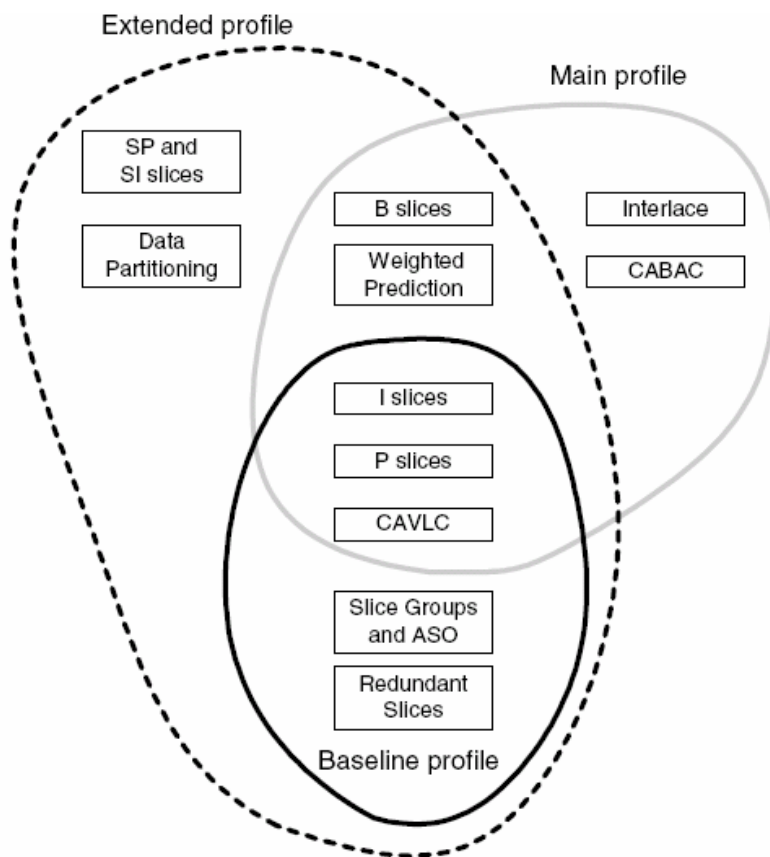


图 6.3 H.264 档次

### 6.3.3 编码数据格式

#### 6.3.3.1 H.264 的视频格式

H.264 支持 4:2:0 的连续或隔行视频的编码和解码，缺省的 4:2:0 的取样格式见图 6.4，图 6.5。

#### 6.3.3.2 H.264 的编码格式

制订 H.264 的主要目标有二个：

- 1) 高的视频压缩比，当初提出的指标是比 H.263，MPEG-4，约为它们的 2 倍，现在都已基本实现；
- 2) 良好的网络亲和性，即可适用于各种传输网络。

为此，H.264 的功能分为两层，即视频编码层（VCL）和网络提取层（NAL，Network Abstraction Layer）。VCL 数据即编码处理的输出，它表示被压缩编码后的视频数据序列。在 VCL 数据传输或存储之前，这些编码的 VCL 数据，先被映射或封装进 NAL 单元中。

每个 NAL 单元包括一个原始字节序列负荷（RBSP）、一组对应于视频编码数据的 NAL 头信息。NAL 单元序列的结构见图 6.6

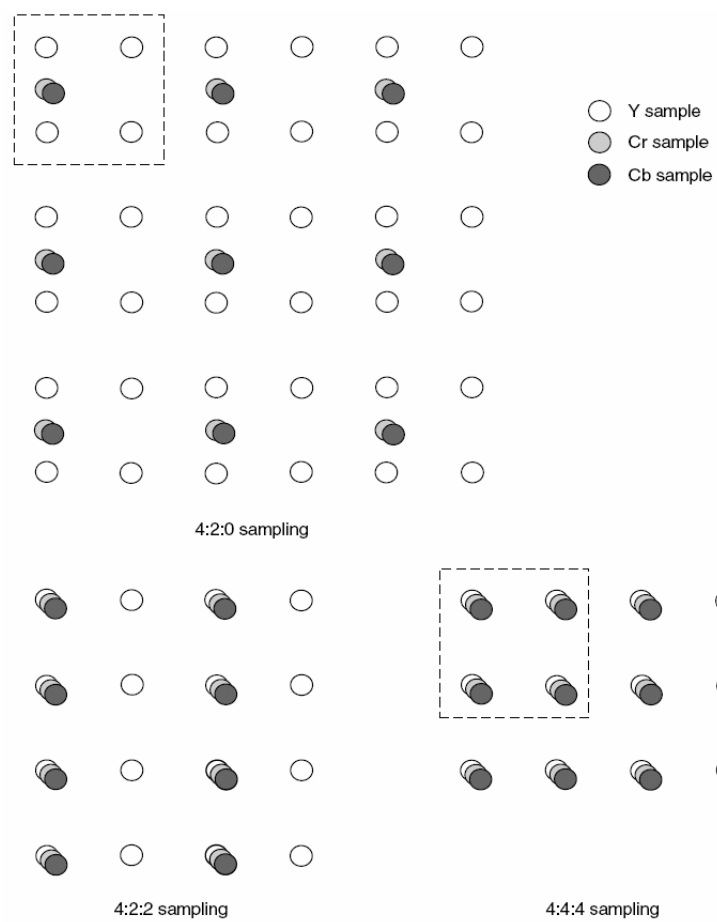


图 6.4 连续视频取样格式

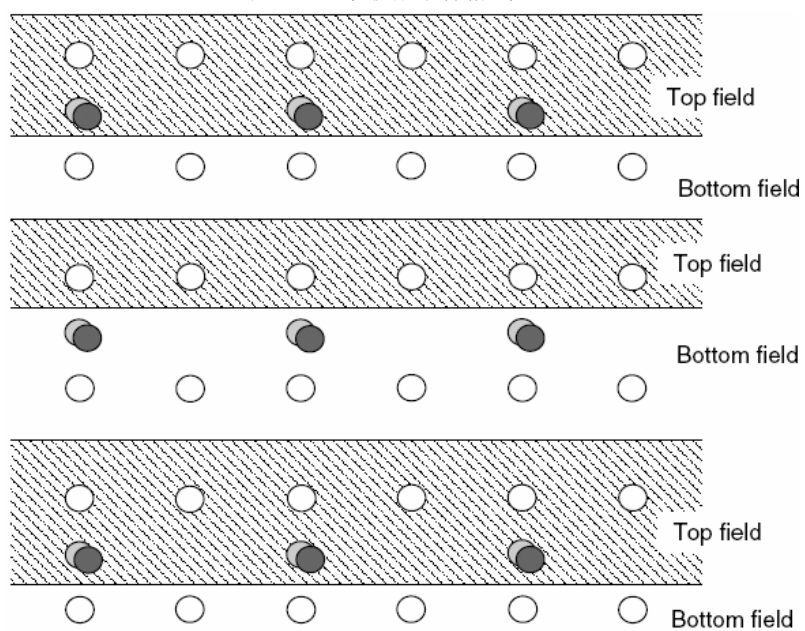


图 6.5 隔行视频取样格式



图 6.6 NAL 单元序列



6.3.4 参数图像

为了提高预测精度，H.264 编码器可从一组前面或后面已编码图像中选出一个或两个与当前最匹配的图像作为帧间编码间的参数图像，这样一来，复杂度大为增加，但多次比较的结果，使匹配后的预测精度显著改进。H.264 中最多可从 15 个参数图像中进行选择，选出最佳的匹配图像。

对于 P 片中帧间编码宏块和宏块分割的预测可从表“0”中选择参数图像；对于 B 片中的帧间编码宏块和宏块分割的预测，可从表“0”和“1”中选择参考图像。

6.3.5 片和片组

现在再详细讨论关于片的结构

6.3.5.1 片

一个视频图像可编码成一个或更多个片，每片包含整数个宏块（MB），即每片至少一个 MB，最多时每片包含整个图像的宏块。总之，一幅图像中每片的宏块数不一定固定。

设片的目的是为了限制误码的扩散和传输，应使编码片相互间是独立的。某片的预测不能以其它片中的宏块为参考图像，这样某一片中的预测误差才不会传播到其它片中去。

编码片共有 5 种不同类型，除已讲过的 I 片、P 片、B 片外，还有 SP 片和 SI 片。其中 SP（切换 P）是用于不同编码流之间的切换；它包含 P 和/或 I 宏块。它是扩展档次中必须具有的切换，它包含了一种特殊类型的编码宏块，叫做 SI 宏块，SI 也是扩展档次中的必备功能。

片的句法结构见图 6.7，其中片头规定了片的类型，该片属于哪个图像，有关的参考图像等，片的数据包含一系列的编码 MB，和/或跳编码（不编码）数据。每个 MB 包含头单元（见表 6.1）和残差数据。

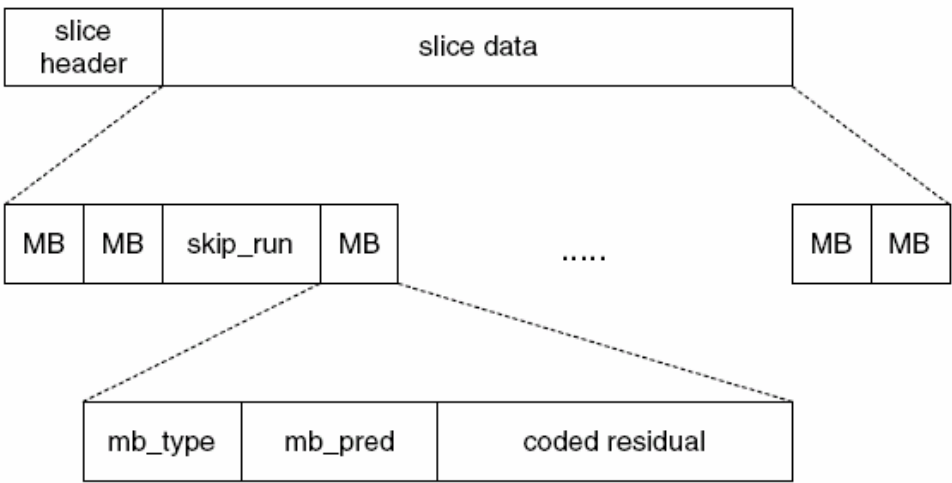


图 6.7 片的句法结构

表 6.1 宏块的句法单元

mb_type	确定该 MB 是帧内或帧间（P 或 B）编码模式，确定该 MB 分割的尺寸
mb_pred	确定帧内预测模式（帧内宏块）确定表 0 或表 1 参考图像，和每一宏块分割的差分编码的运动矢量（帧间宏块，除 8×8 宏块分割的帧内 MB）
sub_mb_pred	（只对 8×8MB 分割的帧内 MB）确定每一子宏块的子宏块分割，每一宏块分割的表 0 和/或表 1 的参考图像；每一宏块子分割的差分编码运动矢量。
coded_block_pattern	指出哪个 8×8 块（亮度和彩色）包含编码变换系数
mb_qp_delta	量化参数的改变值
residual	预测后对应于残差图像取样的编码变换系数

### 6.3.5.2 片组

片组是一个编码图像中若干 MB 的一个子集，它可包含一个或若干个片。

在一个片组中，每片的 MB 按光栅扫描次序被编码，如果每幅图像仅取一个片组，则该图像中所有的 MB 均按光栅扫描次序被编码（除非使用 ASO，即任意的片次序，即一个编码帧中的片之后可跟随任一解码程序的片）。

还有一种片组，叫灵活宏块次序（FMO），它可用灵活的方法，把编码 MB 序列映射到解码图像中 MB 的分配用 MB 到片组之间的映射来确定，它表示每一个 MB 属于哪个片组。表 6.2 为 MB 到片组的各种映射类型。

表 6.2 MB 到片组的映射

类型	名称	描述
0	交错	MB 游程被依次分配给每一块组（图 6.8）
1	散乱	每一片组中的 MB 被分散在整个图像中（图 6.9）
2	前景和背景	例见图 6.10
3	Box-out	从帧的中心开始，产生一个箱子，其 MB 属于片组 0，其它 MB 属于片组（图 6.11）
4	光栅扫描	片组 0 包含按光栅扫描次序从顶-左的所有 MB，其余 MB 属片组 1（图 6.11）
5	手绢	片组 0 包含从顶-左垂直扫描次序的 MB，其余 MB 属片组 1（图 6.11）
6	显式	每一 Mbslice_group_id,用于指明它的片组（即 MB 映射完全是用户定义的）

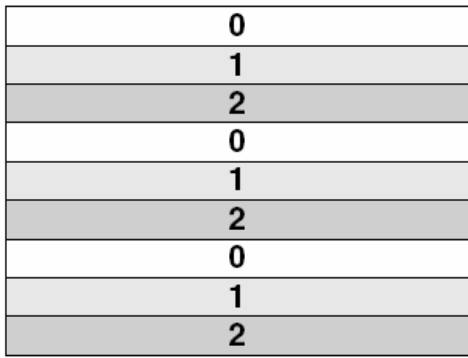


图 6.8 交错型片组

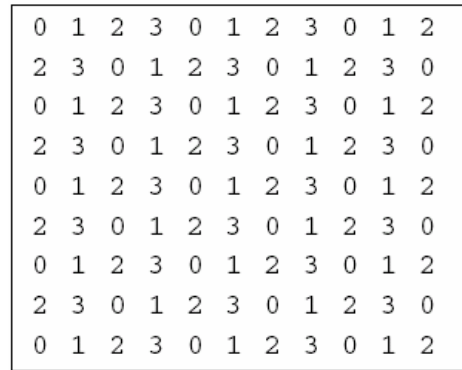


图 6.9 散乱型片组

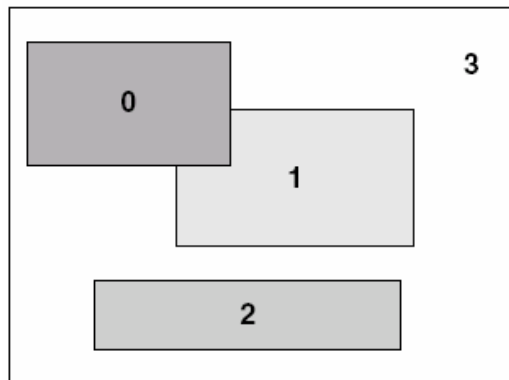


图 6.10 前景和背景型片组

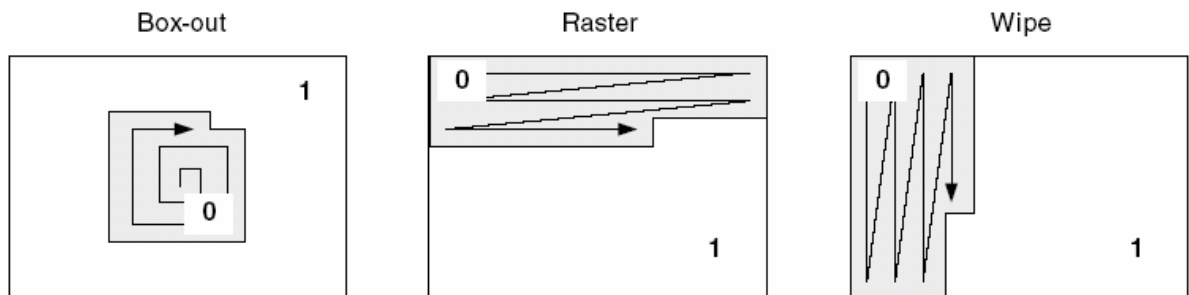


图 6.11 片组

## 6.4 帧内预测

在帧内预测模式中，预测块 P 是基于已编码重建块和当前块形成的。对亮度像素而言，P 块用于  $4 \times 4$  子块或者  $16 \times 16$  宏块的相关操作。 $4 \times 4$  亮度子块有 9 种可选预测模式，独立预测每一个  $4 \times 4$  亮度子块，适用于带有大量细节的图像编码； $16 \times 16$  亮度块有 4 种预测模式，预测整个  $16 \times 16$  亮度块，适用于平坦区域图像编码；色度块也有 4 种预测模式，类似于  $16 \times 16$  亮度块预测模式。编码器通常选择使 P 块和编码块之间差异最小的预测模式。

此外，还有一种帧内编码模式称为 **I\_PCM 编码模式**。该模式下，编码器直接传输图像像素值，而不经预测和变换。在一些特殊的情况下，特别是图像内容不规则或者量化参数非常低时该模式比起“常规操作”（帧内预测—变换—量化—熵编码）效率更高。I\_PCM 模式用于以下目的：

- 1) 允许编码器精确的表示像素值



- 2) 提供表示不规则图像内容的准确值，而不引起重大的数据量增加。
- 3) 严格限制宏块解码比特数，但不损害编码效率。

在以往 H.263+、MPEG-4 等视频压缩编码标准中，帧内编码被引入变换域。H.264 帧内编码则参考预测块左方或者上方的已编码块的邻近像素点，被引入空间域。但是，如果参考预测块是帧间编码宏块，该预测会因参考块的运动补偿引起误码扩散。所以，参考块通常选取帧内编码的邻近块。

如图 6.12 和图 6.13 所示，图 6.12 是采用帧内模式编码的 QCIF 视频帧，其块或者宏块是通过邻近已编码像素预测而得。图 6.13 给出了根据最佳模式而得的预测亮度 P 帧，该模式使得编码信息量最小。



图 6.12 QCIF 帧

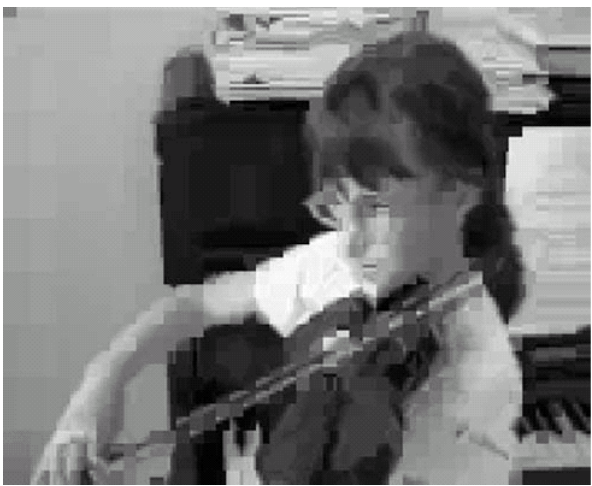


图 6.13 预测帧（帧内预测）

### 6.4.1 4×4 亮度预测模式

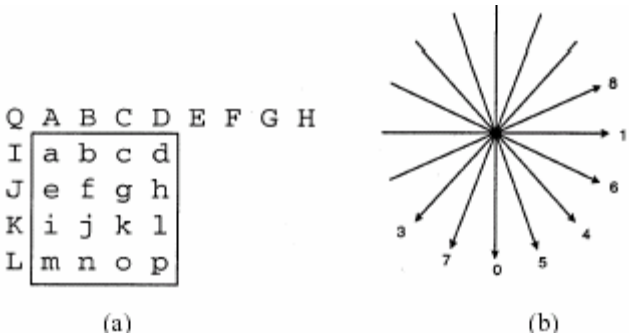


图 6.14 a)利用像素 A-Q 对方块中 a-p 像素进行帧内 4×4 预测  
b)帧内 4×4 预测的 8 个预测方向

如图 6.14 所示，4×4 亮度块的上方和左方像素 A~Q 为已编码和重构像素，用作编解码器中的预测参考像素。a~p 为待预测像素，利用 A~Q 值和 9 种模式实现。其中模式 2(DC 预测)根据 A~Q 中已编码像素预测，而其余模式只有在所需预测像素全部提供才能使用。图 6.15 箭头表明了每种模式预测方向。对模式 3~8，预测像素由 A~Q 加权平均而得。例如，模式 4 中， $p = \text{round}(B/4 + C/2 + D/4)$ 。表 6.5.1 给出了这 9 种模式的描述。

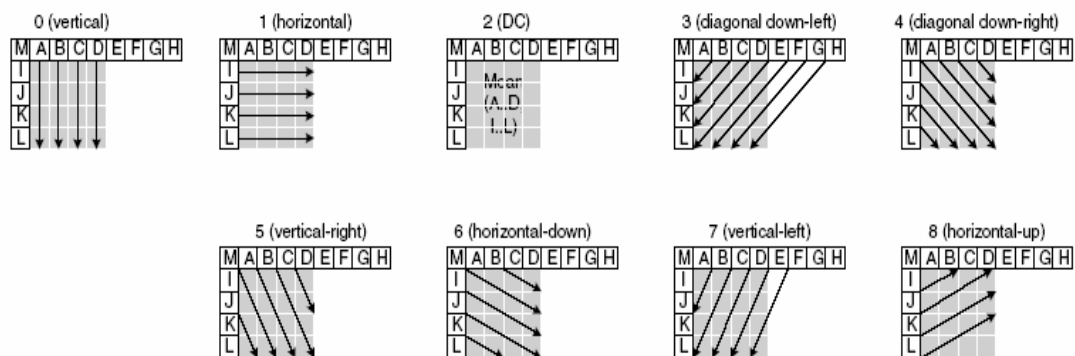


图 6.15 4×4 亮度块预测模式

表 6.3 预测模式描述

模式	描 述
模式 0（垂直）	由 A、B、C、D 垂直推出相应像素值
模式 1（水平）	由 I、J、K、L 水平推出相应像素值
模式 2（DC）	由 A~D 及 I~L 平均值推出所有像素值
模式 3（下左对角线）	由 $45^0$ 方向像素内插得出相应像素值
模式 4（下右对角线）	由 $45^0$ 方向像素内插得出相应像素值
模式 5（右垂直）	由 $26.6^0$ 方向像素值内插得出相应像素值
模式 6（下水平）	由 $26.6^0$ 方向像素值内插得出相应像素值
模式 7（左垂直）	由 $26.6^0$ 方向像素值内插得出相应像素值
模式 8（上水平）	由 $26.6^0$ 方向像素值内插得出相应像素值

举例：图 6.13 所示的 4×4 块的 9 种预测模式计算产生图 6.16 所示的相应预测块（SAE 定义了每种预测的预测误差）。该例中，与当前块的最匹配的模型为模式 8，因为该模式 SAE 最小且最接近于原始 4×4 块。

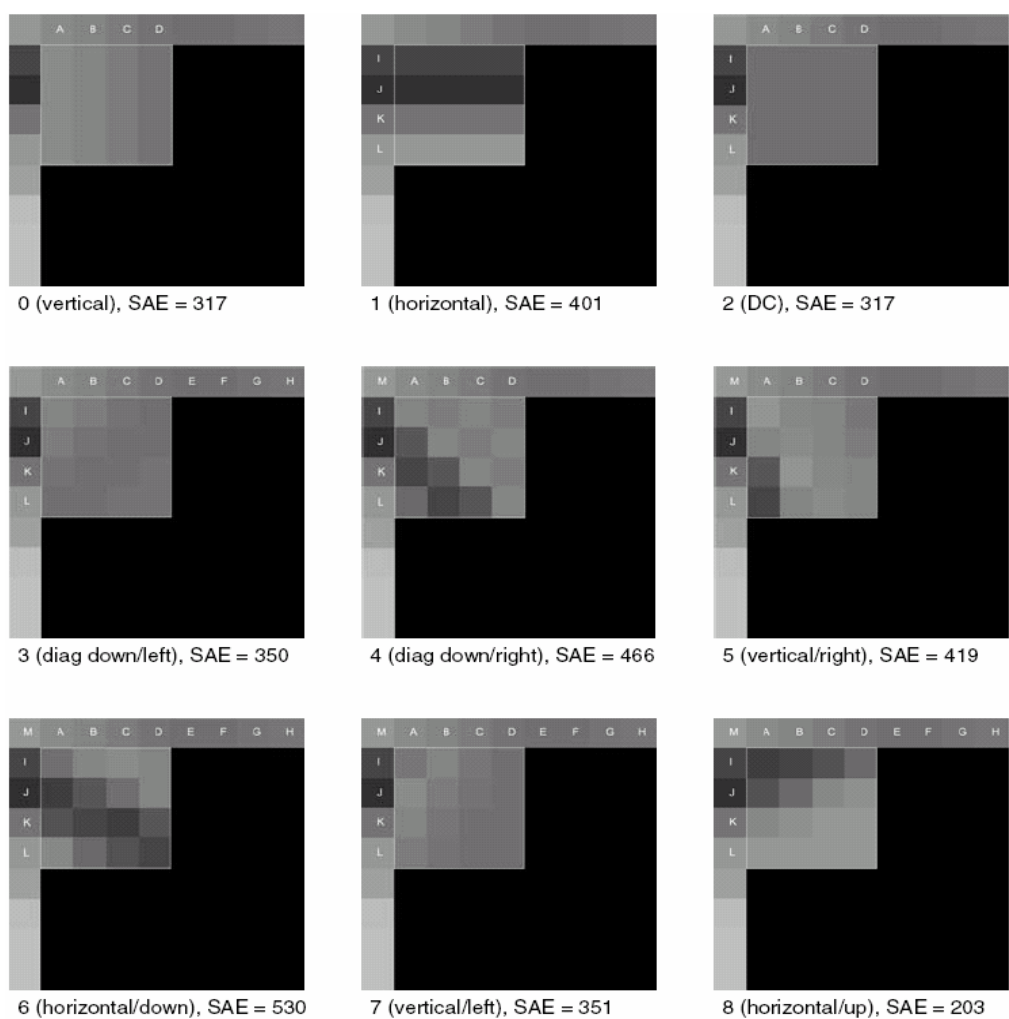


图 6.16 预测块（亮度 4×4）

## 6.4.2 16×16 亮度预测模式

宏块的全部 16×16 亮度成分可以整体预测，有 4 种预测模式，如表 6.4 和图 6.17 所示。

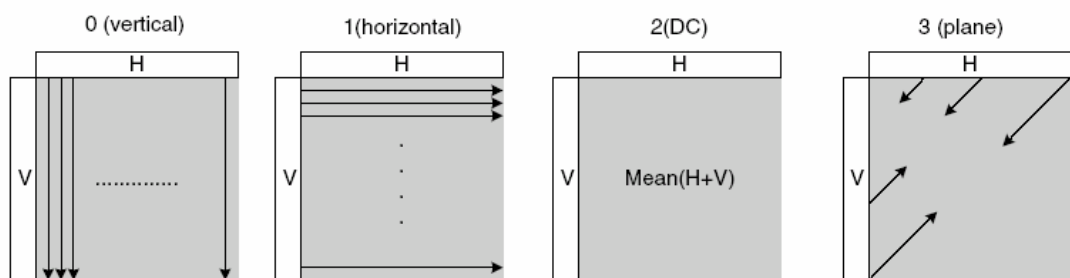


图 6.17 16×16 预测模式

表 6.4 16×16 预测模式

模式	描 述
模式 0（垂直）	由上边像素推出相应像素值
模式 1（水平）	由左边像素推出相应像素值
模式 2（DC）	由上边和左边像素平均值推出相应像素值
模式 3（平面）	利用线形“plane”函数及左、上像素推出相应像素值，适用于亮度变化平缓区域

举例：图 6.18 给出了一个左上方像素已编码的亮度宏块。图 6.19 给出了 4 种预测模式预测结果。其中模式 3 最匹配原始宏块。帧内 16×16 模式适用于图像平坦区域预测。

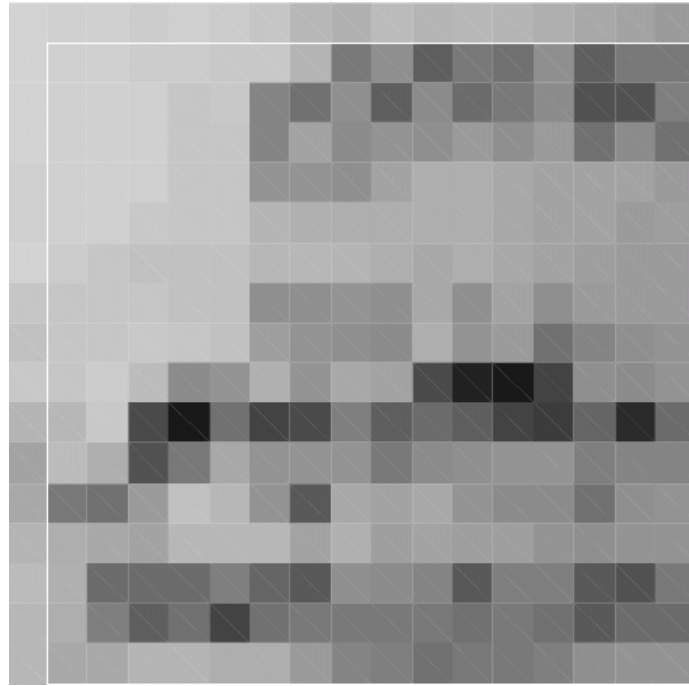


图 6.18 16×16 宏块

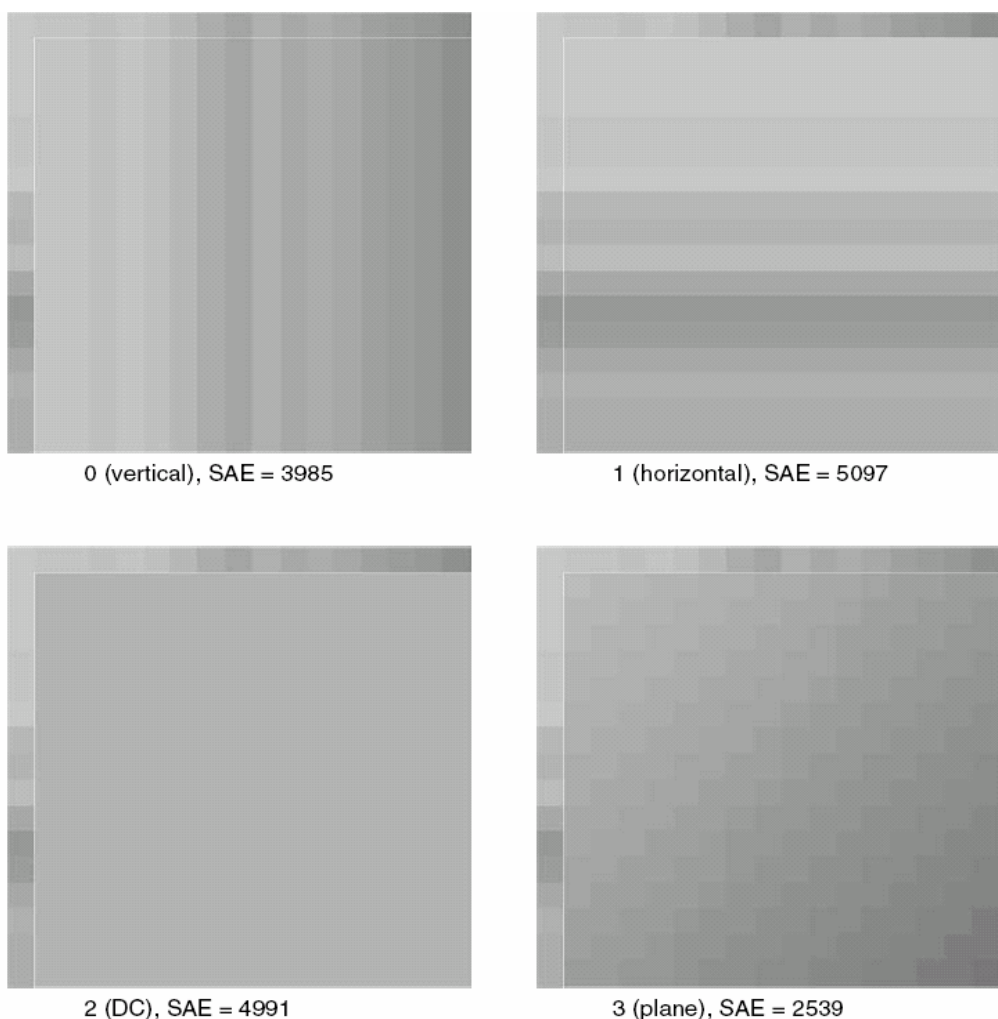


图 6.19 帧内 16×16 预测块

### 6.4.3 8×8 色度块预测模式

每个帧内编码宏块的 8×8 色度成分由已编码左上方色度像素预测而得，两种色度成分常用同一种预测模式。4 种预测模式类似于帧内 16×16 预测的 4 种预测模式，只是模式编号不同。其中 DC（模式 0）、水平（模式 1）、垂直（模式 2）、平面（模式 3）。

### 6.4.4 信号化帧内预测模式

每个 4×4 块帧内预测模式必须转变成信号传给解码器。该信息可能需大量比特表示，但邻近块的帧内模式通常是相关的。例如，A、B、E 分别为左边、上边和当前块，如果 A 和 B 预测模式为模式 1，E 的最佳预测模式很可能也为模式 1。所以通常利用这种关联性信号化 4×4 帧内模式。

对每个当前块 E，编码器和解码器计算最可能预测模式和 A、B 预测模式的较小者。如果这些相邻块不被提供（当前片外或者非帧内 4×4 模式），相应值 A 或 B 置 2（DC 预测模式）。

编码器分配每个 4×4 块一个标志 `prev_intra4×4_pred_mode`。该标志置 1 时，使用最可能预测模式；置 0 时，使用参数 `rem_intra4×4_pred_mode` 来指明模式变化。`rem_intra4×4_pred_mode` 小于当前最可能模式时，预测模式选 `rem_intra4×4_pred_mode`。否则预测模式为 `rem_intra4×4_pred_mode+1`。`rem_intra4×4_pred_mode` 值为 0~7。

举例：块 A 和 B 分别用模式 3 和模式 1 预测。最可能模式则为 1，`prev_intra4×4_pred_mode` 置

0, rem\_intra4×4\_pred\_mode 被传送。取决于 rem\_intra4×4\_pred\_mode 的值，表 6.5 所示的 8 种预测模式中的一种被选定。

表 6.5 预测模式选择（最可能模式为 1）

rem_intra4×4_pred_mode	prediction mode for block C
0	0
1	2
2	3
3	4
4	5
5	6
6	7
7	8

帧内 16×16 亮度和色度预测模式在宏块头中指明。  
这里需说明的是，包括帧内预测的所有预测都不能跨片边界预测，每片必须独立编解码。

可见，帧内预测以绝对误差和（SAE）为标准选取最佳预测模式，使预测帧更加接近原始帧，减少了相互间的差异，去除时间上的数据冗余，提高了编码的压缩效率。帧内预测中，块或宏块利用已编码并重建的块作为参考，进行预测。具体编程时，编码器通过计算并比较各种模式下的 SAE，选取 SAE 值最小的模式作为最佳预测模式，并将该模式信息化，同时传送至解码端，以供正确解码。其具体操作可见第八章相关部分。

6.5 帧间预测

H.264 帧间预测是利用已编码视频帧/场和基于块的运动补偿的预测模式。与以往标准帧间预测的区别在于块尺寸范围更广（从 16×16 到 4×4）、亚像素运动矢量的使用（亮度采用 1/4 像素精度 MV）及多参考帧的运用等等。

下文将重点讨论基本（Baseline）档次支持的 P 片帧间预测工具及主要（Main）和扩展（Extended）档次支持的 B 片和加权预测等帧间预测工具。SP/SI 的内容将在后面的章节中有详细的阐述。

6.5.1 树状结构运动补偿

每个宏块（16×16 像素）可以 4 种方式分割：一个 16×16，两个 16×8，两个 8×16，四个 8×8。其运动补偿也相应有四种。而 8×8 模式的每个子宏块还可以四种方式分割：一个 8×8，两个 4×8 或两个 8×4 及 4 个 4×4。这些分割和子宏块大大提高了各宏块之间的关联性。这种分割下的运动补偿则称为树状结构运动补偿。



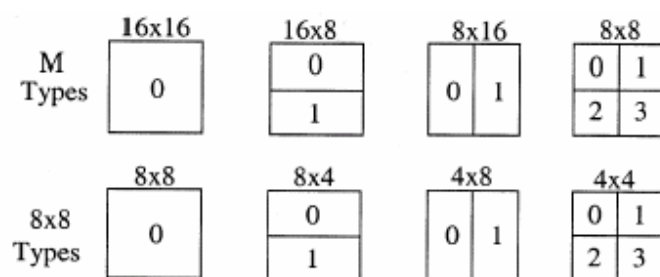


图 6.20 宏块及子宏块分割

每个分割或子宏块都有一个独立的运动补偿。每个 MV 必须被编码、传输，分割的选择也需编码到压缩比特流中。对大的分割尺寸而言，MV 选择和分割类型只需少量的比特，但运动补偿残差在多细节区域能量将非常高。小尺寸分割运动补偿残差能量低，但需要较多的比特表征 MV 和分割选择。分割尺寸的选择影响了压缩性能。整体而言，大的分割尺寸适合平坦区域，而小尺寸适合多细节区域。

宏块的色度成分（Cr 和 Cb）则为相应亮度的一半（水平和垂直各一半）。色度块采用和亮度块同样的分割模式，只是尺寸减半（水平和垂直方向都减半）。例如，8×16 的亮度块相应色度块尺寸为 4×8，8×4 亮度块相应色度块尺寸为 4×2 等等。色度块的 MV 也是通过相应亮度 MV 水平和垂直分量减半而得。

举例：图 6.21 显示了一个残差帧（没有进行运动补偿）。H.264 编码器为帧的每个部分选择了最佳分割尺寸，使传输信息量最小，并将选择的分割加到残差帧上。在帧变化小的区域（残差显示灰色），选择 16×16 分割；多运动区域（残差显示黑色或白色），选择更有效的小的尺寸。

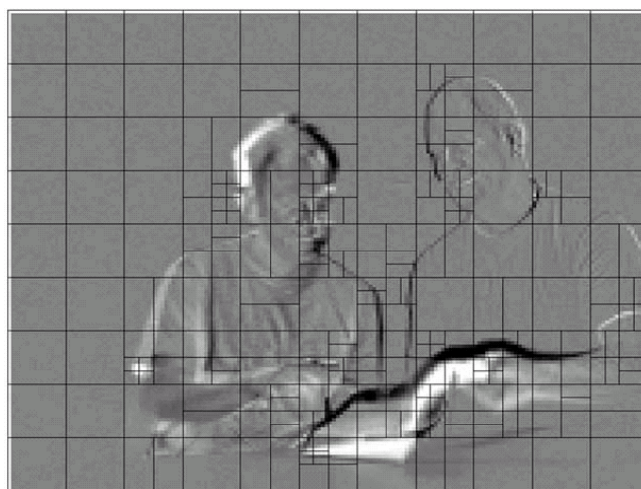


图 6.21 残差帧

## 6.5.2 运动矢量

帧间编码宏块的每个分割或者子宏块都是从参考图像某一相同尺寸区域预测而得。两者之间的差异（MV）对亮度成分采用 1/4 像素精度，色度 1/8 像素精度。亚像素位置的亮度和色度像素并不存在于参考图像中，需利用邻近已编码点进行内插而得。图 6.6.3 中，当前帧的 4×4 块通过邻近参考图像相应区域预测。如果 MV 的垂直和水平分量为整数，参考块相应像素实际存在（灰色点）。如果其中一个或两个为分数，预测像素（灰色点）通过参考帧中相应像素（白色点）内插获得。

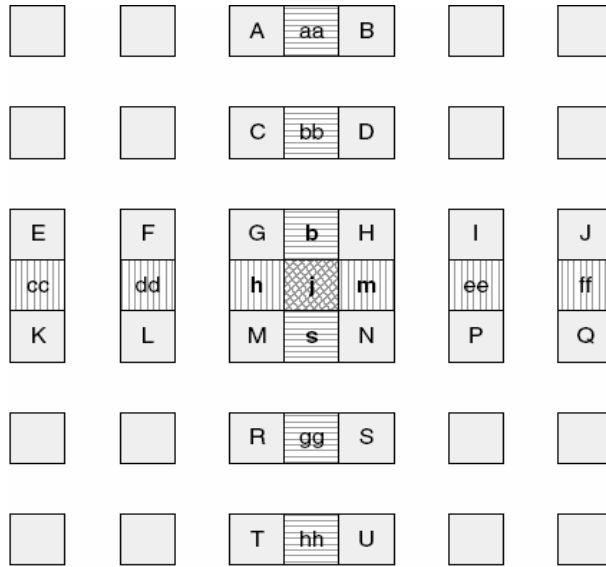


图 6.22 亮度半像素位置内插

### 内插像素生成:

首先生成参考图像亮度成分半像素像素。半像素点（如 b,h,m）通过对相应整像素点进行 6 抽头滤波得出，权重为  $(1/32, -5/32, 5/8, 5/8, -5/32, 1/32)$ 。b 计算如下：

$$b = \text{round}((E - 5F + 20G + 20H - 5I + J) / 32) \quad (6.1)$$

类似的，h 由 A、C、G、M、R、T 滤波得出。一旦邻近（垂直或水平方向）整像素点的所有像素都计算出，剩余的半像素点便可以通过对 6 个垂直或水平方向的半像素点滤波而得。例如，j 由 cc, dd, h,m,ee,ff 滤波得出。这里说明的是，6 抽头滤波器比较复杂，但可明显改善运动补偿性能。

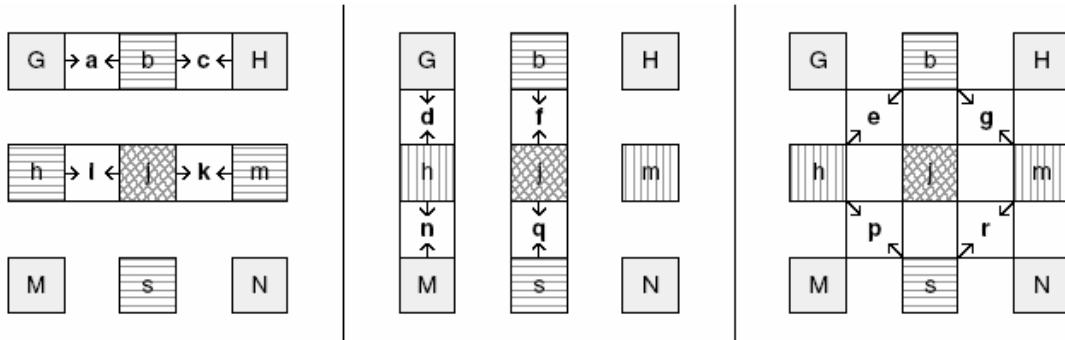


图 6.23 亮度 1/4 像素内插

半像素点计算出来以后，1/4 像素点就可通过线性内插得出，如图 6.23 所示。1/4 像素点（如 a, c, i, k, d, f, n, q）由邻近像素内插而得，如

$$a = \text{round}((G + b) / 2) \quad (6.2)$$

剩余 1/4 像素点（p, r）由一对对角半像素点线性内插得出。如，e 由 b 和 h 获得。

相应地，色度像素需要 1/8 精度地 MV，也同样通过整像素地线性内插得出，如图 6.24 所示。



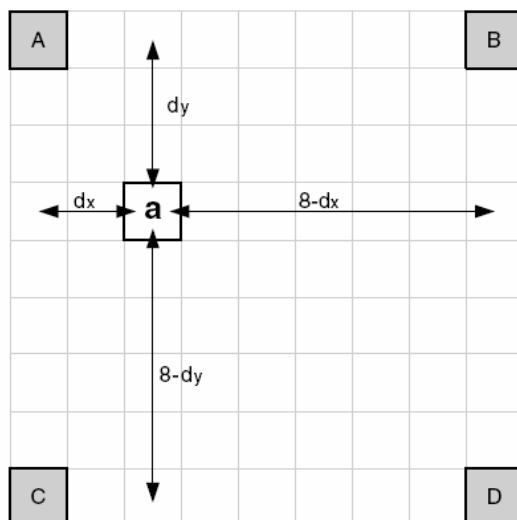


图 6.24 色度 1/8 像素内插

其中,

$$\mathbf{a} = \text{round}([(8 - d_x) \cdot (8 - d_y)A + d_x \cdot (8 - d_y)B + (8 - d_x) \cdot d_yC + d_x \cdot d_yD]/64) \quad (6.3)$$

当  $d_x=2$ ,  $d_y=3$  时,

$$\mathbf{a} = \text{round}[(30A + 10B + 18C + 6D)/64] \quad (6.4)$$

### 6.5.3 MV 预测

每个分割 MV 的编码需要相当数目的比特,特别是使用小分割尺寸时。为减少传输比特数,可利用邻近分割的 MV 较强的相关性, MV 可由邻近已编码分割的 MV 预测而得。预测矢量  $MV_p$  基于已计算 MV 和 MVD (预测与当前的差异) 并被编码和传送。 $MV_p$  则取决于运动补偿尺寸和邻近 MV 的有无。

E 为当前宏块或宏块分割子宏块。A、B、C 分别为 E 的左、上、右上方的三个相对应块。如果 E 的左边不止一个分割,取其中最上的一个为 A;上方不止一个分割时,取最左边一个为 B。图 6.25 显示所有分割有相同尺寸时的邻近分割选择。图 6.26 给出了不同尺寸时临近分割的选择。

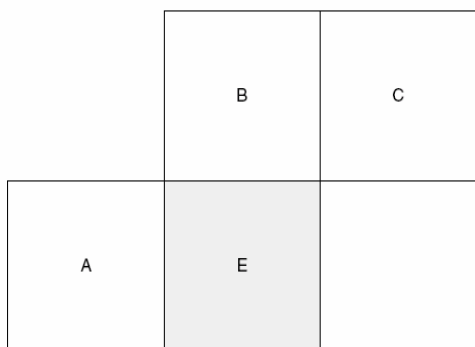


图 6.25 当前和邻近分割 (相同尺寸)

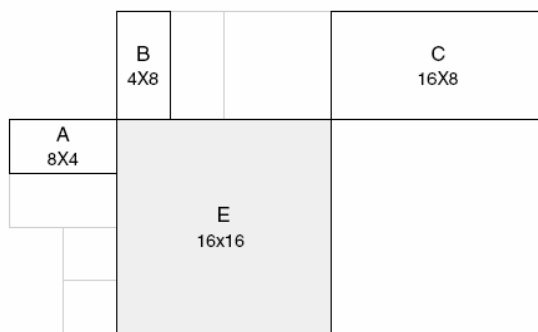


图 6.26 当前和邻近分割 (不同尺寸)

其中:

- 1) 传输分割不包括  $16 \times 8$  和  $8 \times 16$  时,  $MV_p$  为 A、B、C 分割 MV 的中值;
- 2)  $16 \times 8$  分割, 上面部分  $MV_p$  由 B 预测, 下面部分  $MV_p$  由 A 预测;
- 3)  $8 \times 16$  分割, 左面部分  $MV_p$  由 A 预测, 右面部分  $MV_p$  由 C 预测;
- 4) 跳跃宏块 (skipped MB), 同 1)。

如果图 6.26 所示的已传送块不提供时（如在当前片外），MV<sub>p</sub> 选择需重新调整。在解码端，MV<sub>p</sub> 以相同方式形成并加到 MVD 上。对跳跃宏块，不存在 MVD，运动补偿宏块也由 MV 直接生成。

## 6.5.4 B 片预测

B 片中的帧间编码宏块的每个子块都是由一个或两个参考图像预测而得。该参考图像在当前图像的前面或者后面。参考图像存储于编解码器中，其选择有多种方式。图 6.27 显示了三种方式：一个前向和一个后向的（类似于 MPEG 的 B 图像预测）；两个前向；两个后向。

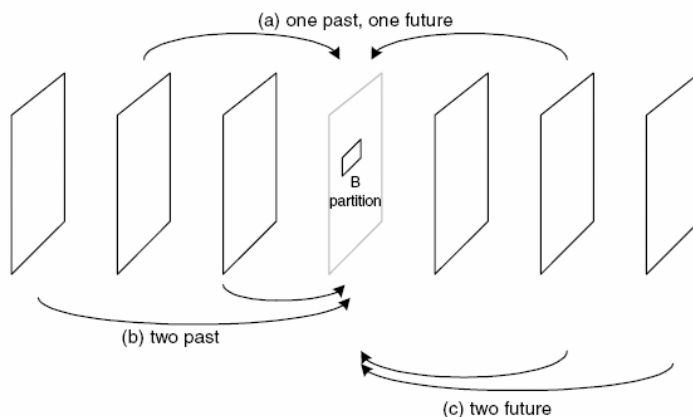


图 6.27 分割预测举例

### （1）参考图像

B 片用到了两个已编码图像列表：list0 和 list1，包括短期和长期图像两种。这两个列表都可包含前向和后向的已编码图像（按显示顺序排列）。其中，

List0：最近前向图像（基于 POC）标为 index0，接着是其余前向图像（POC 递增顺序），及后向图像（从当前图像 POC 递增顺序）。

List1：最近后向图像标为 index0，接着是其余后向图像（POC 递增顺序），及前向图像（从当前图像 POC 递增顺序）。

举例：一个 H.264 解码器存储了 6 幅短期参考图像。其 POC 分别为：123，125，126，128，129 和 130。当前图像为 127。所有 6 幅短期参考图像在 list0 和 list1 中都标为“用作参考”，如表 6.6 所示。

表 6.6 短期缓冲索引

Index	List 0	List 1
0	126	128
1	125	129
2	123	130
3	128	126
4	129	125
5	130	123

选择的缓冲索引作为 Exp-Golomb codeword 发送。最有效的参考索引（codeword 最小）便是 index0（例如：前向编码图像在 list0 中，后向编码图像在 list1 中）。

### （2）预测模式选择

B 片的预测方式包括：宏块分割方式、双向选择方式、参考列表选择方式等等。具体说，B 片

中宏块分割可由多种预测方式中的一种实现，如直接模式、利用 list0 的运动补偿模式、利用 list1 的运动补偿模式或者利用 list0 和 list1 的双向运动补偿模式。每个分割可选择各自的不同的预测模式(如表 6.7 所示)。如果 8×8 分割被使用，每个 8×8 分割所选则的模式适用于分割中的所有亚分割。图 6.28 给出了例子，左边的两个 16×8 分割分别使用 List0 和双向预测模式，而右边的 4 个 8×8 分割分别采用直接、list0、list1 和双向预测四种模式。

表 6.7 B 片宏块预测选则

分割	选 择
16×16	直接、list0、list1、双向
16×8/8×16	list0、list1、双向（每个分割独立选择）
8×8	直接、list0、list1、双向（每个分割独立选择）

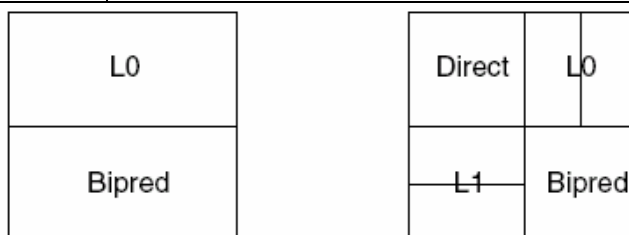


图 6.28 B 片中分割预测模式举例

### (3) 双向预测

双向预测中，参考块（与当前分割或亚分割同尺寸）是由 list0 和 list1 的参考图像推出的。从 list0 和 list1 分别得出两个运动补偿参考区域（需要两个 MV），而预测块的像素取 list0 和 list1 相应像素的平均值。当不用加权预测时，用下列等式：

$$\text{pred}(i,j) = (\text{pred0}(i,j) + \text{pred1}(i,j) + 1) >> 1 \quad (6.5)$$

其中，pred0(i,j)和 pred1(i,j)为由 list0 和 list1 参考帧推出的预测像素，pred(i,j)为双向预测像素。计算出每个预测像素后，运动补偿残差通过当前宏块像素减 pred(i,j)而得。

举例：一宏块用 B\_Bi\_16×16 模式预测。图 6.29 和图 6.30 分别给出了基于 list0 和 list1 参考图像的运动补偿参考区域。图 6.31 给出了根据者两个参考区域的双向预测。

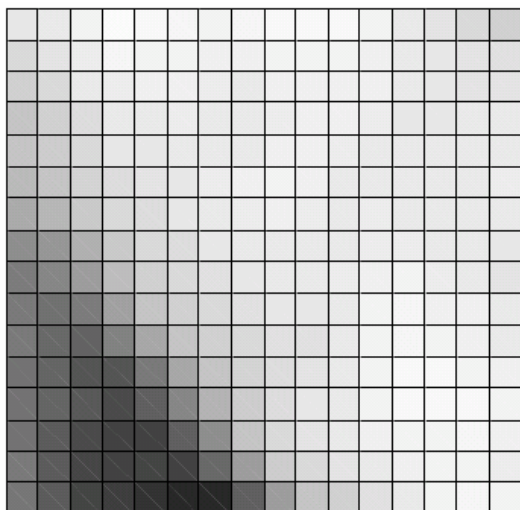


图 6.29 参考区域(list0)

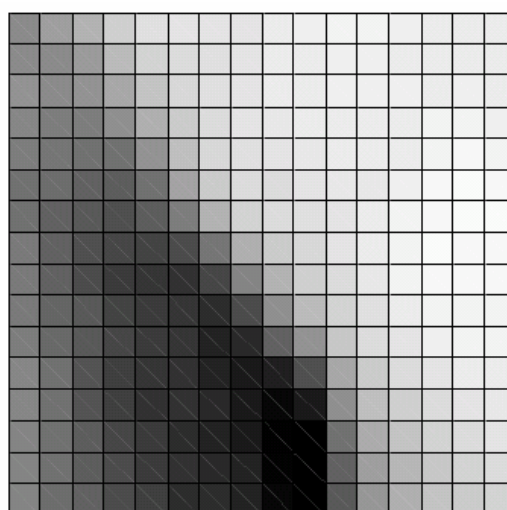


图 6.30 参考区域(list1)

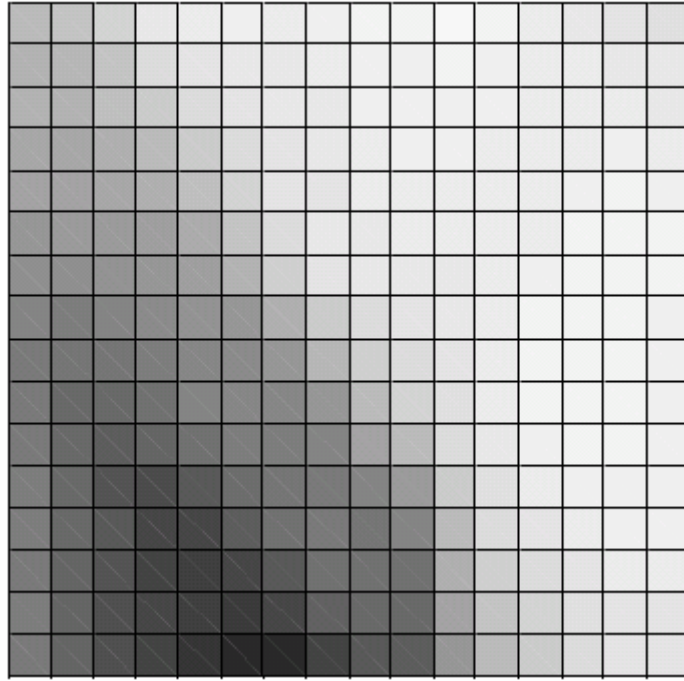


图 6.31 双向预测（无加权）

双向预测宏块或块中的 list0 和 list1 矢量根据邻近相同方向的 MV 预测而得。

#### （4）直接预测

直接预测模式编码的 B 片宏块或宏块分割不传送 MV。相反，解码器计算基于已编码 MV 的 list0 和 list1MV 并计算出解码残差像素的双向预测运动补偿。B 片中的 skipped 宏块便由解码器用直接模式重建而得。

片头会指明将用时间还是空间方式计算直接模式或其分割的矢量。

在空间模式中，list0 和 list1 预测矢量计算如下：

如果第一幅 list1 参考图像的 co-located MB 或分割有一个 MV 幅度上小于 $\pm 1/2$  亮度像素，其一个或两个预测矢量置为 0；否则预测 list0 和 list1 矢量用以计算双向运动补偿。

在时间模式中，计算步骤如下：

- 1) 找出 list1 图像 co-located MB 或分割相应的 list0 参考图像。该 list0 参考作为当前 MB 或分割的 list0 参考；
- 2) 找出 list1 图像 co-located MB 或分割相应的 list0MV；
- 3) 计算当前图像和 list1 图像的 POC 的 MV，作为新的 list1 MV1；
- 4) 计算当前图像和 list0 图像的 POC 的 MV，作为新的 list0 MV0。

这些模式在预测参考宏块或分割不提供或帧内编码等情况下需作出调整。

举例：当前宏块 list1 参考在当前帧两幅图像后出现，如图 6.32 所示。List1 参考 co-located MB 有一 MV (+2.5, +5)，指向 list0 参考图像（出现于当前图像 3 幅图像前）。解码器分别计算指向 list1 和 list0 的 MV1 (-1, -2) 和 MV0 (+1.5, +3)。

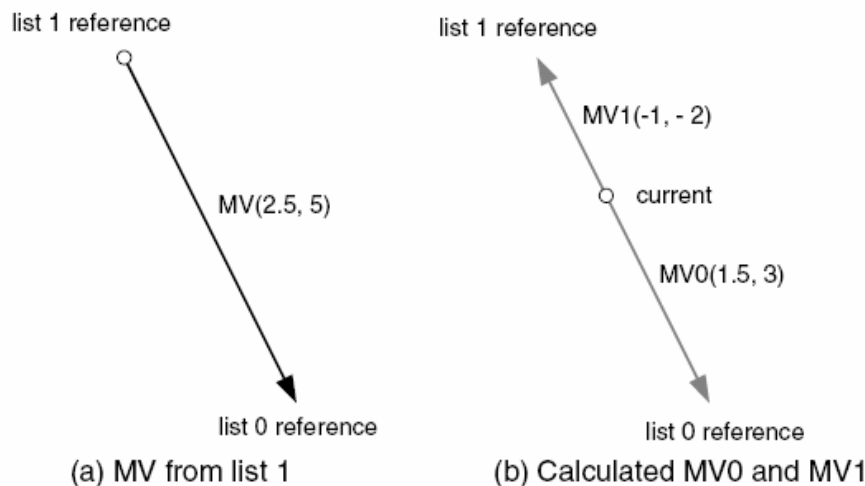


图 6.32 时间直接运动矢量举例

### 6.5.5 加权预测

加权预测是一种用来修正 P 或 B 片中运动补偿预测像素方法。H.264 中由 3 中加权预测类型：

- 1) P 片宏块“explicit”加权预测；
- 2) B 片宏块“explicit”加权预测；
- 3) B 片宏块“implicit”加权预测；

每个预测像素  $\text{pred0}(i,j)$  和  $\text{pred1}(i,j)$  在运动补偿之前通过加权系数  $\omega_0$  和  $\omega_1$  修正。在“explicit”类型中，加权系数由编码器决定并在片头中传输。在“implicit”类型中，系数  $\omega_0$  和  $\omega_1$  由相应 list0 和 list1 参考图像的时间位置推出。大的系数用于时间上接近当前图像的情况，小的则用于时间上远离当前图像的情况。

可见，H.264 采用树状结构的运动补偿技术，提高了预测能力。特别是，小块预测提高了模型处理更好的运动能够描述的能力，产生更好的图像质量。H.264 运动向量的精度提高到 1/4 像素（亮度），运动补偿算法的预测能力得到进一步提高。H.264 还提供多参考帧可选模式，这将产生更好的视频质量和效率更高的视频编码。相对于 1 帧参考，5 个参考帧可以节约 5%~10% 的比特率，且有助于比特流的恢复。当然，并不是说参考帧越多越好，经实验，考虑到缓冲区的能力和编码器的效率，**目前一般都选取 3~5 个参考帧。**

## 6.6 H.264 的 SP/SI 帧技术（SP 片或 SI 宏块的 P 宏块）

当前视频编码标准主要包括三种的帧类型：I 帧、P 帧和 B 帧。随着 H.264/AVC 为了顺应视频流的带宽自适应性和抗误码性能的要求，又定义了两种新的帧类型：SP 帧和 SI 帧。

SP 帧编码的基本原理同 P 帧类似，仍是基于帧间预测的运动补偿预测编码，两者之间的差异在于 SP 帧能够参照不同参考帧重构出相同的图像帧。充分利用这一特性，SP 帧可取代了 I 帧，广泛应用于流间切换（bitstream switching）、拼接（splicing）、随机接入（random access）、快进快退（fast forward, fast backward）以及错误恢复（error recovery）等应用中，同时大大降低了码率的开销。与 SP 帧相对应，SI 帧则是基于帧内预测编码技术，其重构图像和对 SP 的重构图像完全相同。

SP 帧的编码效率尽管略低于 P 帧，但却远远高于 I 帧，大大改善了 H.264 的网络亲和性，支持

灵活的流媒体服务应用，具有很强的抗误码性能，适应在噪声干扰大、丢包率高的无线信道中传输。

### 6.6.1 SP/SI 帧的应用

视频流传输已经成为了“尽力而为”的因特网和无线网络的基本应用，而网络传输条件是时变的。正是为了满足视频流切换的需求，H.264 提出了 SP/SI 帧技术，从而解决视频流应用中终端用户可用带宽不断变化、不同内容节目拼接时数据量的激增、快进快退以及错误恢复等问题。

#### ● 流间切换

视频服务器应该可以根据网络条件，调整编码码率，实现带宽的自适应性。这涉及会话业务和流业务两种应用场合。其中会话业务的特点是实时编解码、点对点传输，一般根据解码端的反馈信息，调整码率或帧率；而流业务的特点是非实时编解码，这往往采用预先编码出不同质量和带宽要求的视频码流。

下面主要以流业务的应用为例，众所周知，实现带宽自适应的最好方法是设置多组不同的信源编码参数对同一视频序列分别进行压缩，从而生成适应不同质量和带宽要求的多组相互独立的码流。这样，视频服务器只需在不同的码流间切换，以适应网络有效带宽的不断变化。

设  $\{P_{1,n-1}, P_{1,n}, P_{1,n+1}\}$  和  $\{P_{2,n-1}, P_{2,n}, P_{2,n+1}\}$  分别是同一视频序列采用了不同的信源编码参数编码所得到的两个视频流。由于编码参数不同，两个码流中同一时刻的帧，如  $P_{1,n-1}$  和  $P_{2,n-1}$ ，并不完全一样。假设服务器首先发送视频流  $P_1$ ，到时刻  $n$  再发送视频流  $P_2$ ，则解码端接收到的视频流为  $\{P_{1,n-2}, P_{1,n-1}, P_{2,n}, P_{2,n+1}, P_{2,n+2}\}$ 。在这种情况下，由于接收的  $P_{2,n}$  应使用的参考帧应该是  $P_{2,n-1}$  而不是  $P_{1,n-1}$ ，所以  $P_{2,n}$  帧就不能完全正确地解码。在以往的视频压缩标准中，实现码流间的切换功能时，确保完全正确解码的前提条件是切换帧不得使用当前帧之前的帧信息，即只使用 I 帧。在实际的流业务码流切换中，往往通过周期性地放置 I 帧确实能实现流间切换等功能，但 I 帧的插入势必造成视频流数据量增大，增加传输带宽的要求。

从 SP 帧的特性可知，使用不同的参考帧作预测，也可以得到完全相同的解码帧。这一特点正好适用于流间切换。如图 6.33 所示中，视频流明确的切换点处放置主 SP 帧（Primary SP-frames）— $S_{1,n}$  帧和  $S_{2,n}$  帧。在不同编码参数的视频流间进行切换时，发送与主 SP 帧相对应的辅 SP 帧（Secondary SP-frames）— $S_{12,n}$ 。当然信源编码部分根据需要插入 SP/SI 帧，也适当增加了编码的复杂度。

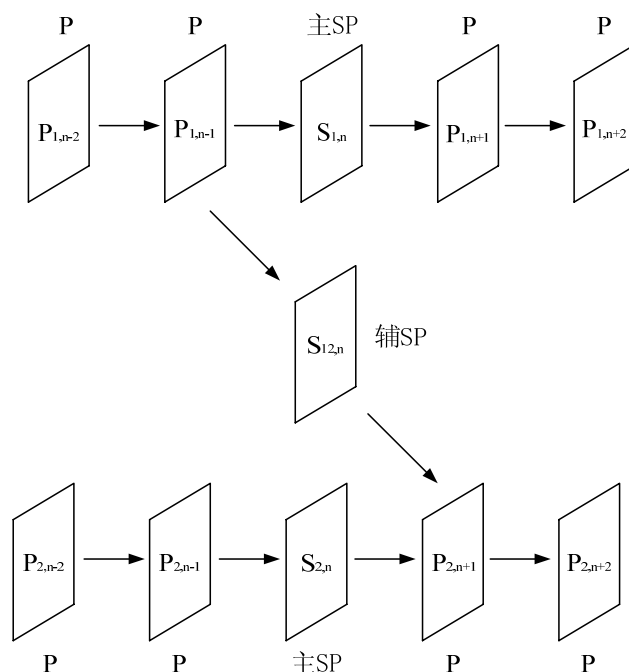


图 6.33 SP 帧进行流间切换

- 拼接（Splicing）与随机接入（Random Access）

上述流间切换的例讨论了同一图像序列、不同编码参数压缩编码的流间切换。然而，实际的流间切换的应用并不单单如此。例如，关注同一事件而处于不同视角的多台摄像机的输出码流间的切换和电视节目中插入广告等，这就涉及到拼接不同图像序列生成码流的问题。如图 6.34 所示，由于各个码流来自于不同的信源，帧间缺乏相关性，切换点处的辅帧如果仍采用帧间预测的辅 SP 帧，编码效率就不会那么有效，而应采用空间预测的 SI 帧—— $SI_{2,n}$  帧。

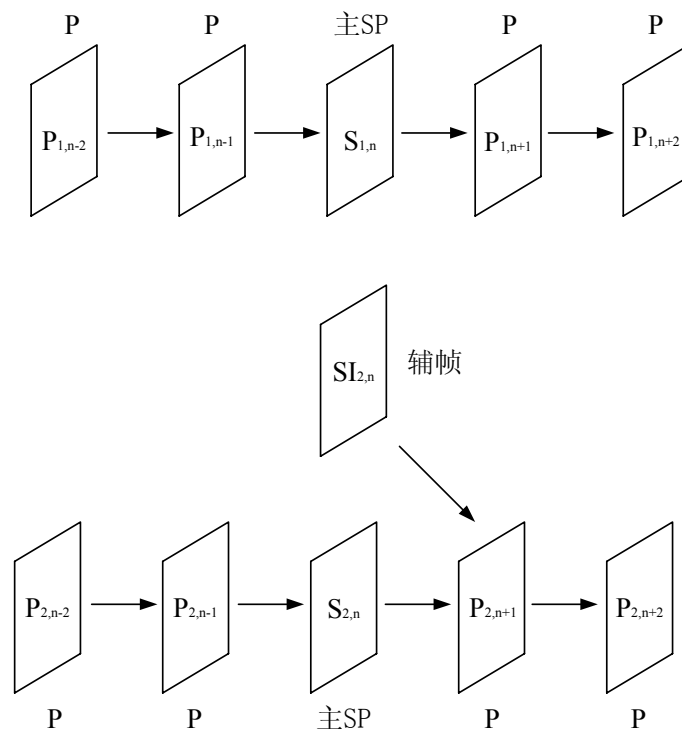


图 6.34 SI 帧进行拼接和随机接入

综上所述，SP 帧与 SI 帧均可用于流间切换。当视频流的内容相同，编码参数不同采用 SP 帧；而当视频流的内容相差很大时，则采用 SI 帧将更加有效。

- 错误恢复

采用不同的参考帧预测，可以获得同一帧的多个 SP 帧，利用这种特性可以增强错误恢复的能力。例如，如图 6.35 所示，正在进行视频流传输的比特流中的一个帧  $P_{l,n-1}$  无法正确解码，得到用户端反馈的错误报告后，服务器就可以发送其后最邻近主 SP 帧的一个辅 SP 帧—— $SI_{l2,n}$ ，以避免错误影响更多后续帧， $SI_{l2,n}$  帧的参考帧是已经正确解码的帧。当然，用户端也可以使用辅 SI 帧—— $SI_{l1,n}$  来实现相同的功能。



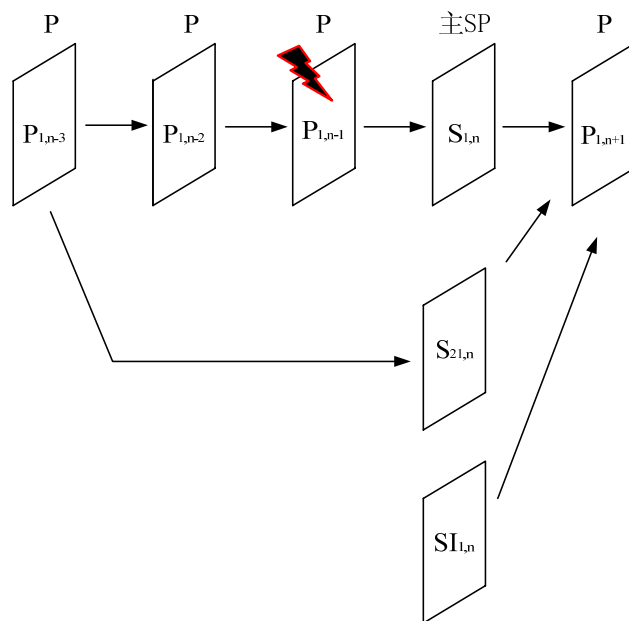


图 6.35 SP 帧进行错误恢复

### 6.6.2 SP/SI 帧的基本原理

从上述的应用可知，SP 帧分为主 SP 帧（Primary SP-Frame）和辅 SP 帧（Secondary SP-Frame）。其中，前者的参考帧和当前编码帧属于同一个码流，而后者则不属于同一个码流。与此同时，如图 6.36 所示，主 SP 帧作为切换插入点，不切换时，码流进行正常的编码传输；而切换时，辅 SP 帧取代主 SP 帧进行传输。其中， $\Delta$ ：编码器的输入； $\bigcirc$ ：解码器的输出。

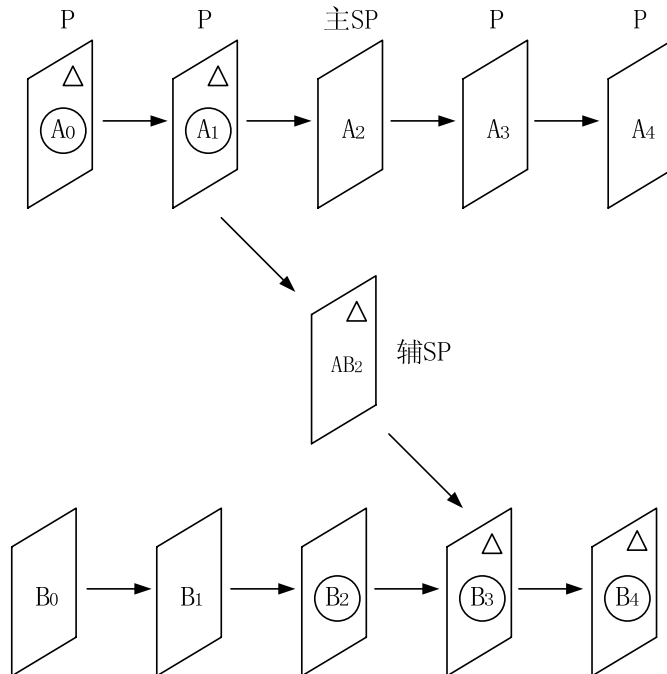


图 6.36 流间切换中 SP 帧编码顺序图

下面描述 SP 帧实现的功能原理。SI 帧的功能实现与 SP 帧相类似，都能恢复出相同的解码图像帧，只是前者利用的是帧间预测，而后者利用的则是帧内预测。

#### ● 主 SP 帧的编码过程

主 SP 帧的编码过程如图 6.37 所示。从结构上看，与传统 P 帧编码不同之处在于：变换处理提



到了求差值，即前者的预测残差是变换系数的差值。改进的目的是在于使得主辅两种 SP 帧的编码器能相结合。主 SP 帧的编码的具体过程如下：预测预测块  $P(x,y)$  是利用原始图像和已重建帧进行运动补偿预测得到的。对预测块  $P(x,y)$  和原始图像中相对应的块分别进行正向变换，然后用量化参数  $SPQP$  对预测块  $P(x,y)$  的变换系数进行量化和反量化，得到系数  $d_{pred}$ 。从原始图像中相对应的变换系数中减去  $d_{pred}$  就得到预测残差  $c_{err}$ ，然后将  $c_{err}$  用量化参数  $PQP$  进行量化，其结果  $l_{err}$  和运动矢量一起传送到多路复用器。

在这个编码器中，用于对  $c_{rec}$  也就是对预测重构块系数进行量化的量化参数  $SPQP$  与对预测残差系数  $c_{err}$  进行量化的量化参数  $PQP$  没有必要相同。因此可对预测块系数采用不同于对预测残差系数采用的、引入更小失真的量化参数，以使产生的重建误差更小。

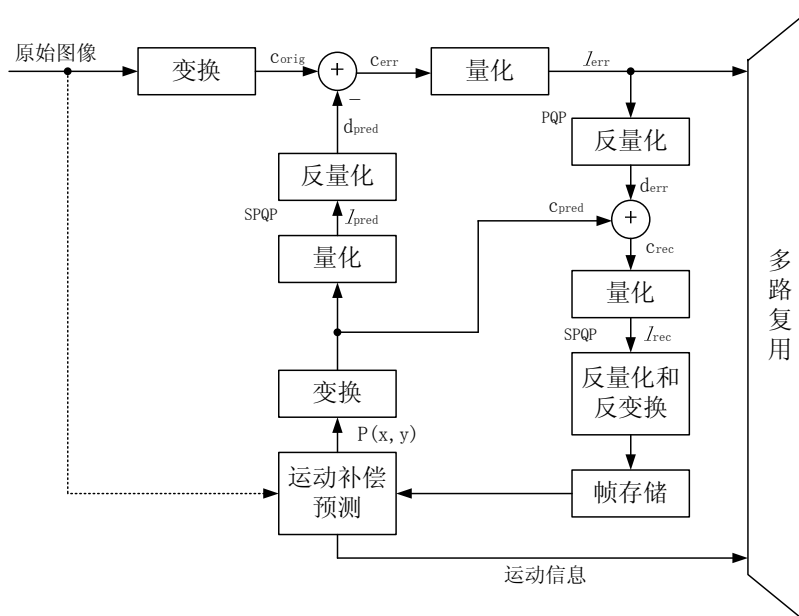


图 6.37 SP 帧的编码原理图

以主 SP 帧  $A_2$  的编码过程为例，如图 6.38 所示。与 P 帧编码过程相比，主 SP 帧的编解码过程有所不同。视频序列中帧  $A_2$  和帧  $A_1'$ （帧  $A_2$  参考帧  $A_1$  的重建图像）分别经过变换处理，得到的变换系数计算出变换系数差值，再进行量化和熵编码，最终得到主 SP 帧的编码。

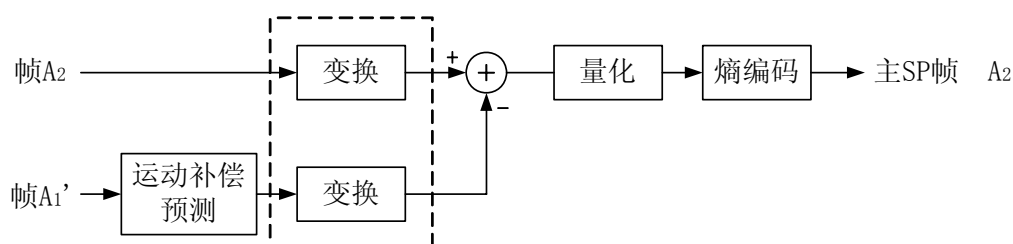


图 6.38 主 SP 帧  $A_2$  的编码过程示意图

### ● 主 SP 帧的解码

图 6.39 描述了主 SP 帧的解码原理图。获得预测块  $P(x,y)$ ，后进行正变换，变换后得到的系数记为  $c_{pred}$ ，将编码端输出的已经量化的预测残差系数  $l_{err}$  采用量化参数  $PQP$  进行反量化，得到的系数记为  $d_{err}$ 。 $d_{err}$  与  $c_{pred}$  的和记为  $c_{rec}$ ，重建系数  $c_{rec}$  采用量化参数  $SPQP$  进行量化和反量化，并对反量化后获得的  $d_{rec}$  进行反变换和滤波即得重建图像。

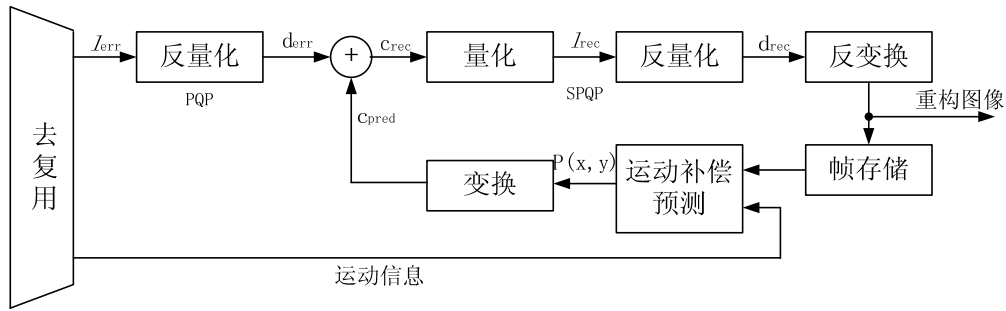


图 6.39 主 SP 帧解码原理图

以主 SP 帧  $A_2$  的解码过程为例，如图 6.40 所示。主 SP 帧  $A_2$  经过熵解码和反量化，和（帧  $A_2$  参考帧  $A_1$  的重建图像）分别经过运动补偿预测和变换处理的结果相加，得到重建图像的变换系数值，再进行反变换，最终得到主 SP 帧的重构图像帧  $A_2'$ 。

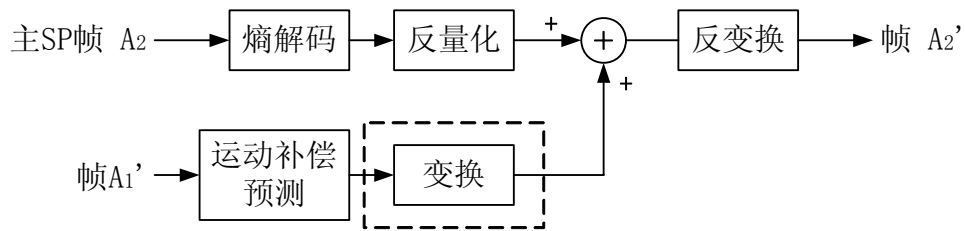


图 6.40 主 SP 帧  $A_2$  的解码过程示意图

#### ● 辅 SP/SI 帧编码过程

设以  $P_1(x, y)$  作为预测帧时，主 SP 帧的重建图像为  $I_c(x, y)$ ， $I_c(x, y)$  由重建系数  $l_{rec}$  做反量化和反变换得到。假设另一预测帧为  $P_2(x, y)$ ，若要得到与主 SP 帧具有相同重建图像  $I_c(x, y)$  的辅 SP 帧，所要做的就是找到新的预测残差系数  $l_{rec,2}$ ，使得利用  $P_2(x, y)$  而不是  $P_1(x, y)$  也能准确地重建图像  $I_c(x, y)$ 。对  $P_2(x, y)$  进行变换并量化，量化后的系数记为  $l_{pred,2}$ ，则辅 SP 帧的预测残差系数  $l_{err,2}$  可按下式计算得到

$$l_{err,2} = l_{rec} - l_{pred,2}$$

然后把计算所得的预测残差系数  $l_{err,2}$  进行无损的熵编码。解码端生成  $P_2(x, y)$  后，对  $P_2(x, y)$  进行变换和量化，得到  $l_{pred,2}$ ，其值与编码器端的值相同。 $l_{pred,2}$  与接收到的预测残差系数  $l_{err,2}$  相加，所得和为  $l_{rec}$ ，对  $l_{rec}$  进行反量化和反变换，就可得到了重建图像  $I_c(x, y)$ ，其与以  $P_1(x, y)$  作为预测帧得主 SP 帧所重建得图像完全相同。辅 SP 帧的编码过程也如图 6.37 所示。

以辅 SP 帧  $AB_2$  的编码过程为例，如图 6.41 所示。

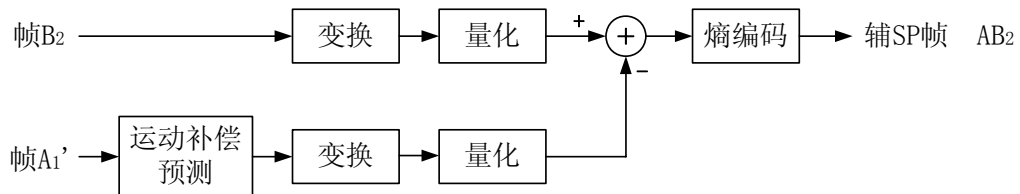


图 6.41 辅 SP 帧  $AB_2$  的编码过程示意图

#### ● 辅 SP/SI 帧的解码

图 6.42 是辅 SP/SI 帧的解码原理图。首先获得预测块  $P(x, y)$ ，对于 SP 帧， $P(x, y)$  是利用编码端输出的运动矢量和参考帧号等信息经过运动补偿预测得到。对于 SI 帧， $P(x, y)$  则是利用编码端输出的帧内预测模式信息通过空间预测得到。然后，对  $P(x, y)$  进行正向变换，并将获得的变换系数量化，记为  $l_{pred}$ 。 $l_{pred}$  和量化后的预测残差系数  $l_{err}$  相加得到已经量化的重建系数  $l_{rec}$ ， $l_{rec}$  量化得到  $d_{rec}$ ， $d_{rec}$  反变换并加以滤波即得重建图像。

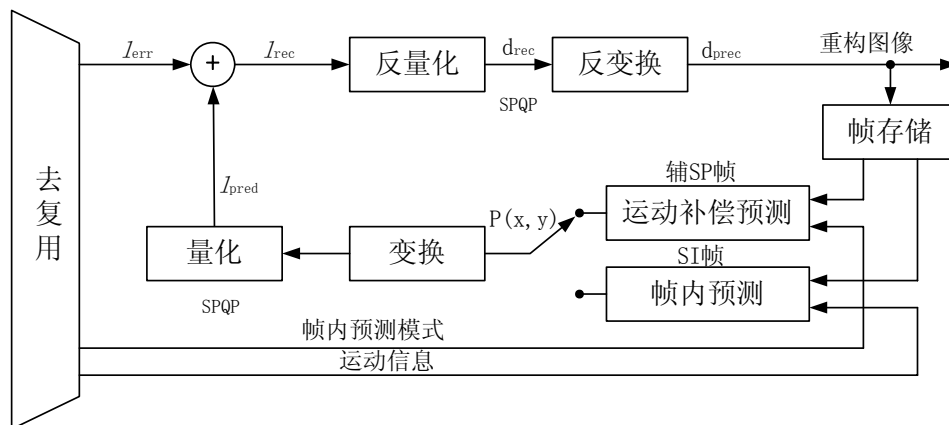
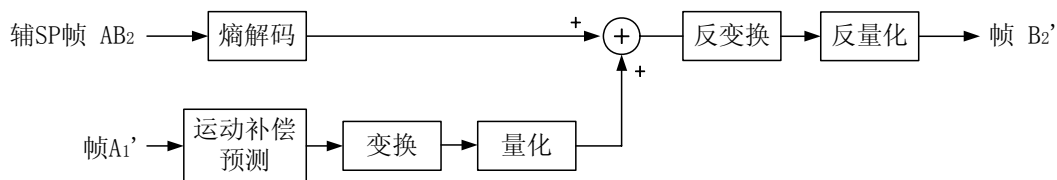


图 6.42 辅 SP/SI 帧的解码原理图

以辅 SP 帧 AB<sub>2</sub> 的编码过程为例, 如图 6.43 所示。

图 6.43 辅 SP 帧 AB<sub>2</sub> 的解码过程示意图

### 6.6.3 实验结果和性能分析

在 TML8.7 参考测试模型基础上, 采用 5 个参考帧的 UVLC 模式, 并选用率失真优化选项。仿真选用的 JVT 测试序列, 帧率为 10 帧/秒。针对 SP 帧的编码效率和序列周期性插入 SP 帧的编码效率分别进行了测试, 实验结果参考了文献 1。

### ● SP 帧的编码效率

图 6.44 给出了 SP 帧、I 帧和 P 帧编码效率的比较。序列的起始帧均按 I 帧编码处理，其余帧分别按 SP 帧、I 帧和 P 帧进行编码。图中 SP 帧编码又给出了三种不同的 SPQP 值：SPQP=PQP、SPQP=3、SPQP=PQP-6。可以看出，SP 帧的编码效率低于 P 帧，当要远远高于 I 帧。选择恰当的 SPQP 值，可以改善 SP 帧的编码效率。其中，SPQP 的值相对于 PQP 的下降，SP 帧的编码效率逐渐提高。当 SPQP=3 时，SP 帧的编码效率非常接近 P 帧的编码效率。

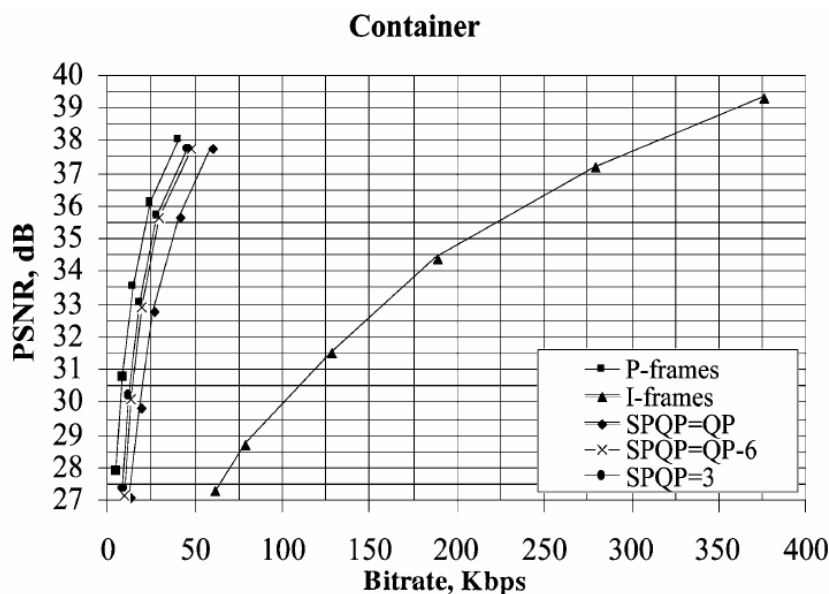


图 6.44 SP 帧、I 帧、P 帧编码效率的比较

#### ● 序列周期性插入 SP 帧的编码性能

如上所述，在流间切换或随机接入等应用中，往往需要在序列中周期性地插入 SP 帧或 I 帧。设定序列的起始帧为 I 帧，每隔一个固定周期为 1s 插入一个 SP 帧或 I 帧，其余帧则仍采用 P 帧。当然只是周期性插入 SP 帧和 I 帧，还不能实现上述功能，当可以给出序列插入 I 帧和 SP 帧后编码性能地的比较。由图 6.45 中可以看出，在实现同样功能的情况下，SP 帧的编码效率比 I 帧高得多，而通过降低 SPQP 值，SP 帧的编码效率将获得进一步改善，逐步逼近 P 帧的编码效率。

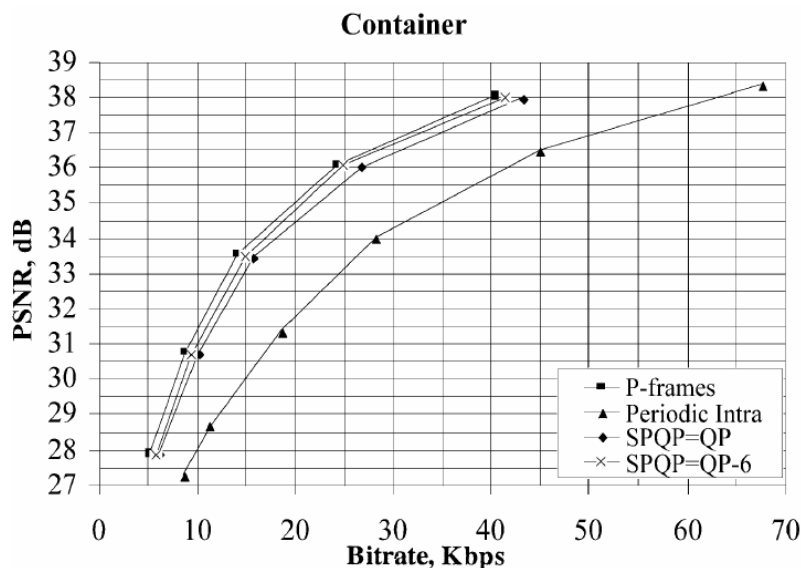


图 6.45 序列周期性插入 SP 帧、I 帧的比较

## 6.7 整数变换与量化

为了进一步节省图像传输码率，需要对图像信号进行压缩，一般方法为去除图像信号中的相关性并减小图像编码的动态范围，通常采用变换编码及量化。变换编码将图像时域信号转换成频域信号，在频域中图像信号能量大部分集中在低频区域，相对时域信号，码率有较大的下降。H.264 对图像或预测残差采用了  $4 \times 4$  整数离散余弦变换技术，避免了以往标准中使用的通用  $8 \times 8$  离散余弦变换逆变换经常出现的失配问题。量化过程根据图像的动态范围大小确定量化参数，既保留图像必要

的细节，又减少码流。

在图像编码中，变换编码和量化从原理上讲是两个独立的过程。但在 H.264 中，将两个过程中的乘法合二为一，并进一步采用整数运算，减少编解码的运算量，提高图像压缩的实时性，这些措施对峰值信噪比（PSNR）的影响很小，一般低于 0.02dB，可不计。H.264 中整数变换及量化具体过程如图 6.46 所示。其中，如果输入块是色度块或帧内 16×16 预测模式的亮度块，则将宏块中各 4×4 块的整数余弦变换的直流分量组合起来再进行 Hadamard 变换，进一步压缩码率。图 6.47 所示的变换量化过程可以用图 6.46 的流程图表示。

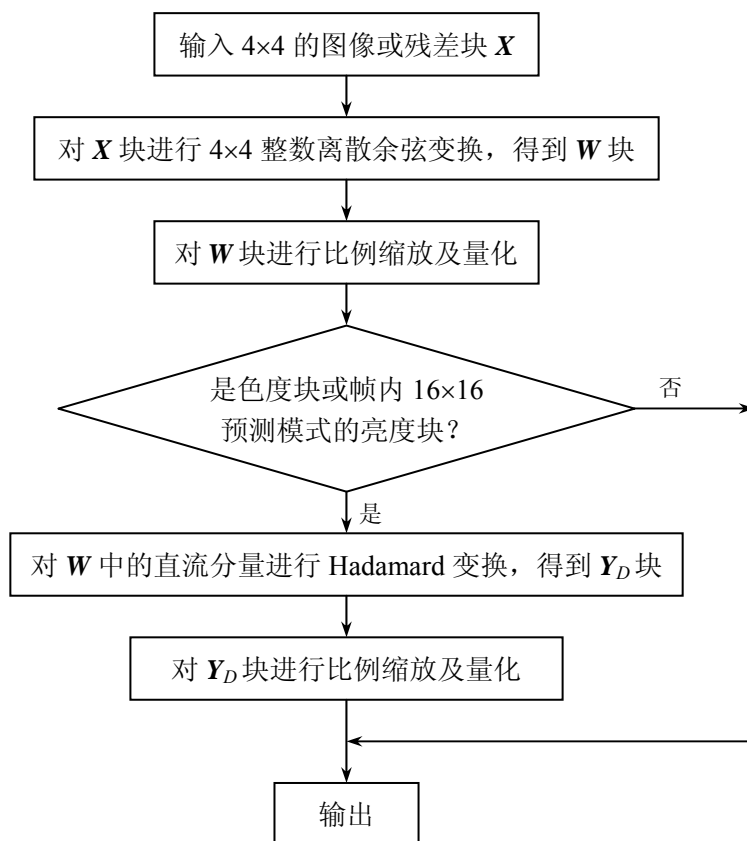


图 6.46 编码器中变换编码及量化过程

本节将介绍图 6.46 和图 6.47 中各部分流程。

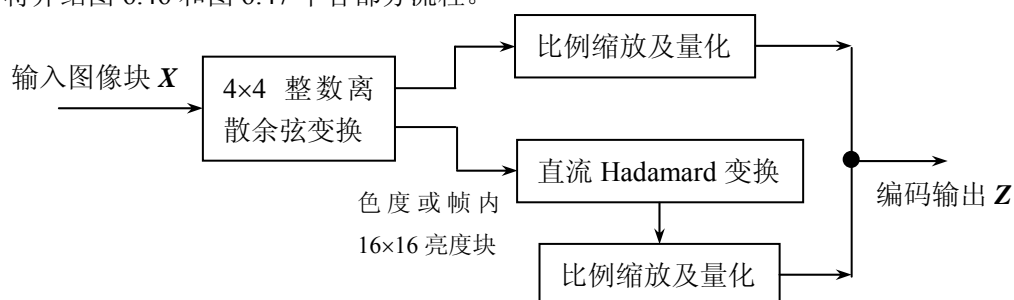


图 6.47 编码变换及量化过程流程图

### 6.7.1 整数变换

一维  $N$  点离散余弦余弦变换（DCT）可以表示为：

$$y_k = C_k \sum_{n=0}^{N-1} x_n \cos \frac{(2n+1)k\pi}{2N} \quad (6.6)$$

其中， $x_n$  是输入时域序列中第  $n$  项， $y_k$  是输出频域序列中第  $k$  项，系数  $C_k$  定义如下：

$$C_k = \begin{cases} \sqrt{\frac{1}{N}} & k = 0 \\ \sqrt{\frac{2}{N}} & k = 1, 2, \dots, N-1 \end{cases} \quad (6.7)$$

每个 DCT 系数  $y_k$  确定信号  $x_n$  在相应频率点上的贡献。最低的系数（即  $k=0$ ）为 DC 系数，代表信号的平均值，也称为直流分量。其它系数称为 AC 系数，与递增的较高频率相联系。

一维  $N$  点离散余弦逆变换（IDCT）可以表示为：

$$x_n = \sum_{k=0}^{N-1} C_k y_k \cos \frac{(2n+1)k\pi}{2N} \quad (6.8)$$

二维  $N \times N$  图像块的 DCT 可以理解为先对图像块的每行进行一维 DCT，然后对经行变换的块的每列再应用一维 DCT。可以表示为：

$$Y_{mn} = C_m C_n \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{ij} \cos \frac{(2j+1)n\pi}{2N} \cos \frac{(2i+1)m\pi}{2N} \quad (6.9)$$

$$X_{ij} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C_m C_n Y_{mn} \cos \frac{(2j+1)n\pi}{2N} \cos \frac{(2i+1)m\pi}{2N} \quad (6.10)$$

其中， $X_{ij}$  是图像块  $\mathbf{X}$  中第  $i$  行第  $j$  列图像或残差值， $Y_{mn}$  是变换结果矩阵  $\mathbf{Y}$  相应频率点上的 DCT 系数。可以用矩阵表示：

$$\mathbf{Y} = \mathbf{A} \mathbf{X} \mathbf{A}^T \quad (6.11)$$

$$\mathbf{X} = \mathbf{A}^T \mathbf{Y} \mathbf{A} \quad (6.12)$$

其中  $N \times N$  变换矩阵  $\mathbf{A}$  中的系数：

$$A_{ij} = C_i \cos \frac{(2j+1)i\pi}{2N} \quad (6.13)$$

H.264 对  $4 \times 4$  的图像块（亮度块或 Cr、Cb 色度块）进行操作，则相应的  $4 \times 4$  DCT 变换矩阵  $\mathbf{A}$  为：

$$\begin{aligned}
\mathbf{A} &= \begin{bmatrix} \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) \\ \sqrt{\frac{1}{2}}\cos(\frac{\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{5\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{7\pi}{8}) \\ \sqrt{\frac{1}{2}}\cos(\frac{2\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{6\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{10\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{14\pi}{8}) \\ \sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{9\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{15\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{21\pi}{8}) \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos(\frac{\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8}) & -\sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8}) & -\sqrt{\frac{1}{2}}\cos(\frac{1\pi}{8}) \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8}) & -\sqrt{\frac{1}{2}}\cos(\frac{\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{\pi}{8}) & -\sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8}) \end{bmatrix}
\end{aligned} \tag{6.14}$$

设  $a = \frac{1}{2}$ ,  $b = \sqrt{\frac{1}{2}}\cos(\frac{\pi}{8})$  及  $c = \sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8})$ , 则:

$$\mathbf{A} = \begin{bmatrix} a & a & a & a \\ b & c & -c & b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \tag{6.15}$$

$\mathbf{A}$  中的  $a$ 、 $b$  和  $c$  是实数，而图像块  $\mathbf{X}$  中的元素是整数。对实数的 DCT，由于在解码端的浮点运算精度问题，会造成解码后的数据的失配，进而引起漂移。H.264 较其它图像编码使用了更多的预测过程，甚至内部编码模式也依赖于空间预测。因此，H.264 对预测漂移是十分敏感的。为此，H.264 对  $4 \times 4$ DCT 中的  $\mathbf{A}$  进行了改造，采用整数 DCT 技术，有效地减少计算量，同时不损失图像准确度。式(6.11)可以等效为：

$$\begin{aligned}
\mathbf{Y} &= (\mathbf{C}\mathbf{X}\mathbf{C}^T) \otimes \mathbf{E} \\
&= \left( \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \mathbf{X} \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}
\end{aligned} \tag{6.16}$$

其中， $d=c/b$  ( $\approx 0.414$ )。符号“ $\otimes$ ”表示  $(\mathbf{C}\mathbf{X}\mathbf{C}^T)$  结果中的每个元素乘以矩阵  $\mathbf{E}$  中对应位置上的系数值的运算。为了简化计算，取  $d$  为 0.5。同时又要保持变换的正交性，对  $b$  进行修正，取  $b = \sqrt{\frac{2}{5}}$ 。

对矩阵  $\mathbf{C}$  中的第 2 行和第 4 行，以及矩阵  $\mathbf{C}^T$  中的第 2 列和第 4 列元素乘以 2，相应地改造矩阵  $\mathbf{E}$  为  $\mathbf{E}_f$ ，以保持式(6.16)成立，得到：

$$Y = (C_f X C_f^T) \otimes E_f$$

$$= \left( \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} X \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \quad (6.17)$$

其中，运算“ $\otimes$ ”对每个矩阵元素只进行一次乘法，同时它将被归纳到量化运算中。这样， $(C_f X C_f^T)$  中只剩下整数的加法、减法和移位（乘以 2）运算。被称为整数 DCT 的式(6.17)与通常 DCT 运算结果近似，但因为 b 和 d 的值有所变化，所以两者结果有差别。例如：输入图像或残差块  $X$  为：

	j=0	1	2	3
i=0	24	20	3	24
1	16	2	1	6
2	23	7	4	5
3	10	20	5	22

其 4×4DCT 输出为：

$$Y = A X A^T = \begin{bmatrix} 48.0 & 10.097 & 17.0 & -9.594 \\ 2.679 & 3.914 & 6.150 & 1.036 \\ 16.0 & -9.277 & -1.0 & -14.558 \\ 6.467 & 6.036 & 1.782 & 1.086 \end{bmatrix} \quad (6.18)$$

而式(6.17)对应的整数 DCT 变换结果为：

$$Y' = (C X C^T) \otimes E = \begin{bmatrix} 48.0 & 10.751 & 17.0 & -8.854 \\ 2.214 & 3.4 & 6.008 & 1.20 \\ 16.0 & -8.222 & -1.0 & -15.179 \\ 6.641 & 6.2 & 2.214 & 1.60 \end{bmatrix} \quad (6.19)$$

两者之差为：

$$Y - Y' = \begin{bmatrix} 0.0 & -0.654 & 0.0 & -0.74 \\ 0.465 & 0.514 & 0.142 & -0.164 \\ 0.0 & -1.055 & 0.0 & 0.621 \\ -0.174 & -0.164 & -0.432 & -0.514 \end{bmatrix} \quad (6.20)$$

H.264 将 DCT 中“ $\otimes E_f$ ”运算的乘法融合到后面的量化过程中，实际的 DCT 输出为：

$$W = C X C^T \quad (6.21)$$

如上如述，可以将式(6.17)的矩阵乘法运算改造成两次一维整数 DCT 变换，例如先对图像或其残差块的每行进行一维整数 DCT，然后对经行变换的块的每列再应用一维整数 DCT。而每次一维整数 DCT 可以采用蝶形快速算法，节省计算时间，如图 6.48 所示。



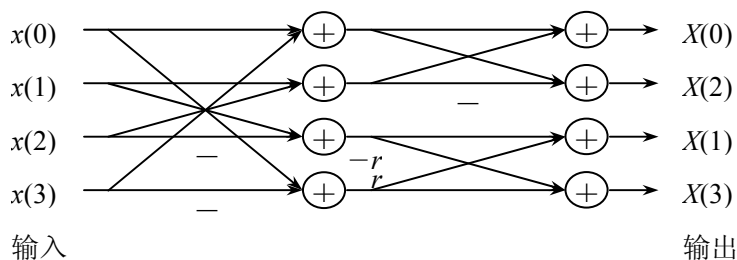


图 6.48 一维快速变换算法

其中， $r=2$ ：整数 DCT 变换； $r=1$ ：Hadamard 变换

## 6.7.2 量化

量化过程在不降低视觉效果的前提下减少图像编码长度，减少视觉恢复中不必要的信息。H.264 采用标量量化技术，它将每个图像样点编码映射成较小的数值。一般标量量化器的原理为：

$$FQ = \text{round}\left(\frac{y}{QP}\right) \quad (6.22)$$

其中， $y$  为输入样本点编码， $QP$  为量化步长， $FQ$  为  $y$  的量化值， $\text{round}()$  为取整函数（其输出为与输入实数最近的整数）。其相反过程，即反量化为：

$$y' = FQ \cdot QP \quad (6.23)$$

在量化和反量化过程中，量化步长  $QP$  决定量化器的编码压缩率及图像精度。如果  $QP$  比较大，则量化值  $FQ$  动态范围较小，其相应的编码长度较小，但反量化时损失较多的图像细节信息；如果  $QP$  比较小，则  $FQ$  动态范围较大，相应的编码长度也较大，但图像细节信息损失较少。编码器根据图像值实际动态范围自动改变  $QP$  值，在编码长度和图像精度之间折衷，达到整体最佳效果。

在 H.264 中，量化步长  $Qstep$  共有 52 个值。如表 6.8 所示。其中  $QP$  是量化参数，是量化步长的序号。当  $QP$  取最小值 0 时代表最精细的量化，当  $QP$  取最大值 51 时代表最粗糙的量化。 $QP$  每增加 6， $Qstep$  增加一倍。应用时可以在这个较宽的量化步长范围根据实际需要灵活选择。对于色度编码，一般使用与亮度编码同样的量化步长。为了避免在较高量化步长时的出现颜色量化人工效应，现在的 H.264 草案把色度的  $QP$  最大值大约限制在亮度  $QP$  最大值的 80% 范围内，最后的 H.264 草案规定，亮度  $QP$  的最大值是 51，而色度  $QP$  的最大值是 39。

表 6.8 H.264 中编解码器的量化步长

QP	Qstep	QP	Qstep	QP	Qstep	QP	Qstep	QP	Qstep
0	0.625	12	2.5	24	10	36	40	48	160
1	0.6875	13	2.75	25	11	37	44	49	176
2	0.8125	14	3.25	26	13	38	52	50	208
3	0.875	15	3.5	27	14	39	56	51	224
4	1	16	4	28	16	40	64		
5	1.125	17	4.5	29	18	41	72		
6	1.25	18	5	30	20	42	80		
7	1.375	19	5.5	31	22	43	88		
8	1.625	20	6.5	32	26	44	104		
9	1.75	21	7	33	28	45	112		
10	2	22	8	34	32	46	128		
11	2.25	23	9	35	36	47	144		

在 H.264 中，量化过程是对前面(6.17)式的 DCT 结果进行操作：

$$Z_{ij} = \text{round}\left(\frac{Y_{ij}}{Qstep}\right) \quad (6.24)$$

其中， $Y_{ij}$  是矩阵  $\mathbf{Y}$  中的转换系数， $Z_{ij}$  是输出的量化系数， $Qstep$  是量化步长。

H.264 量化过程还要同时完成 DCT 变换中“ $\otimes \mathbf{E}_f$ ”乘法运算，它可以表述为：

$$Z_{ij} = \text{round}\left(W_{ij} \frac{PF}{Qstep}\right) \quad (6.25)$$

其中， $W_{ij}$  是矩阵  $\mathbf{W}$  中的转换系数， $PF$  是矩阵  $\mathbf{E}_f$  中的元素，根据样本点在图像块中的位置(i, j)取值：

$$PF = \begin{cases} a^2 & (0,0), (2,0), (0,2) \text{ 或 } (2,2) \\ b^2 / 4 & (1,1), (1,3), (3,1) \text{ 或 } (3,3) \\ ab / 2 & \text{其它情况} \end{cases} \quad (6.26)$$

利用量化步长随量化参数每增加 6 而增加一倍的性质，可以进一步简化计算。设：

$$qbits = 15 + \text{floor}(QP/6) \quad (6.27)$$

$$\text{令 } MF = \frac{PF}{Qstep} 2^{qbits} \quad (6.28)$$

其中， $\text{floor}()$  为取整函数（其输出为不大于输入实数的最大整数）。则式(6.25)可以写成：

$$Z_{ij} = \text{round}\left(W_{ij} \frac{MF}{2^{qbits}}\right) \quad (6.29)$$

这样， $MF$  可以取整数，如表 6.9 所示的量化表。表 6.9 只列出对应  $QP$  值为 0 到 5 的  $MF$  值，对于  $QP$  值大于 5 的情况，只是  $qbits$  值随  $QP$  值每增加 6 而增加 1，而对应的  $MF$  值不变。这样，量化过程则为整数运算，并且可以避免使用除法，并且确保用 16 位算法来处理数据，在没有 PSNR 性能恶化的情况下实现最小的运算复杂度。

表 6.9 H.264 中  $MF$  值

样点位置 QP	(0, 0), (2, 0), (2, 2), (0, 2)	(1, 1), (1, 3), (3, 1), (3, 3)	其它样 点位置
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

具体量化过程的运算为：

$$\begin{aligned} |Z_{ij}| &= (|W_{ij}| \cdot MF + f) \gg qbits \\ \text{sign}(Z_{ij}) &= \text{sign}(W_{ij}) \end{aligned} \quad (6.30)$$

其中，“ $\gg$ ”为右移运算，右移一次完成整数除以 2； $\text{sign}()$  为符号函数； $f$  为偏移量，它的作用是改善恢复图像的视觉效果，例如，对帧内预测图像块  $f$  取  $2^{qbits}/3$ ，对帧间预测图像块  $f$  取  $2^{qbits}/6$ 。

### 6.7.3 DCT 直流系数的变换量化

如果当前处理的图像宏块是色度块或帧内 16×16 预测模式的亮度块，则需要将其中各图像块的 DCT 变换系数矩阵  $\mathbf{W}$  中的直流分量或直流系数  $W_{00}$  按对应图像块顺序排序，组成新的矩阵  $\mathbf{W}_D$ ，再对  $\mathbf{W}_D$  进行 Hadamard 变换及量化。对这种情况，在 6.7.2 节中介绍的量化过程就不需要对各图像块中的  $W_{00}$  单独进行量化。

#### 6.7.3.1 帧内 16×16 预测模式亮度块的直流系数变换量化

16×16 的图像宏块中有 4×4 个 4×4 图像亮度块，所以亮度块的  $\mathbf{W}_D$  为 4×4 矩阵，其组成元素为各图像块 DCT 的直流系数  $W_{00}$ ，这些  $W_{00}$  在  $\mathbf{W}_D$  中的排列顺序为对应图像块在宏块的位置。宏块中亮度块序号与其在宏块中位置的对应关系如图 6.49 所示。其中，16 个大方块代表宏块中 16 个图像亮度块，大方块中的数字为对应图像亮度块在宏块中的序号，大方块左上方的小方块中两个数字代表该图像亮度块在宏块中的行和列。

<div>00</div> 0	<div>01</div> 1	<div>02</div> 4	<div>03</div> 5
<div>10</div> 2	<div>11</div> 3	<div>12</div> 6	<div>13</div> 7
<div>20</div> 8	<div>21</div> 9	<div>22</div> 12	<div>23</div> 13
<div>30</div> 10	<div>31</div> 11	<div>32</div> 14	<div>33</div> 15

图 6.49 宏块中亮度块序号与其在宏块中位置的对应关系

对亮度块  $\mathbf{W}_D$  的 Hadamard 变换为：

$$\mathbf{Y}_D = \left( \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \mathbf{W}_D \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) / 2 \quad (6.31)$$

其中， $\mathbf{Y}_D$  是 Hadamard 变换结果。接着要对  $\mathbf{Y}_D$  再进行量化输出：

$$\begin{aligned} |Z_{D(i,j)}| &= (|Y_{D(i,j)}| \cdot MF_{(0,0)} + 2f) \gg (qbits+1) \\ \text{sign}(Z_{D(i,j)}) &= \text{sign}(W_{D(i,j)}) \end{aligned} \quad (6.32)$$

其中， $MF_{(0,0)}$  是位置为 (0, 0) 的 MF 系数值，其它参数见上节。4×4Hadamard 变换也可以采用快速算法。

### 6.7.3.2 色度块的直流系数变换量化

16×16 的图像宏块中包含图像色度 Cr 及 Cb 块各 2×2 个，所以色度 Cr 或 Cb 块的  $\mathbf{W}_D$  为 2×2 矩阵，其组成元素为各对应图像块色度信号 DCT 的直流系数  $W_{00}$ ，这些  $W_{00}$  在  $\mathbf{W}_D$  中的排列顺序为对应图像块在宏块的位置。宏块中色度块序号与其在宏块中位置的对应关系如图 6.50 所示。其中，4 个大方块代表宏块中 4 个图像色度块，大方块中的数字为对应图像色度块在宏块中的序号，大方块左上方的的小方块中两个数字代表该图像色度块在宏块中的行和列。可以看出，这四个方块的对应关系与图 6.49 中亮度块前四个序号的对应关系一致，有利于软件开发时减少程序代码。

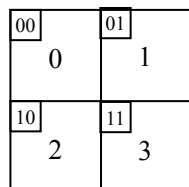


图 6.50 宏块中色度块序号与其在宏块中位置的对应关系

对各色度块  $\mathbf{W}_D$  的 Hadamard 变换为：

$$\mathbf{Y}_D = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \mathbf{W}_D \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (6.33)$$

其中， $\mathbf{Y}_D$  是 Hadamard 变换结果。接着要对  $\mathbf{Y}_D$  再进行量化输出：

$$\begin{aligned} |Z_{D(i,j)}| &= (|Y_{D(i,j)}| \cdot MF_{(0,0)} + 2f) \gg (qbits+1) \\ \text{sign}(Z_{D(i,j)}) &= \text{sign}(W_{D(i,j)}) \end{aligned} \quad (6.34)$$

其中， $MF_{(0,0)}$ 、 $f$ 、 $qbits$  等定义同亮度块部分。

## 6.8 CAVLC(基于上下文自适应的可变长编码)

### 6.8.1 熵编码的基本原理

熵编码是无损压缩编码方法，它生成的码流可以经解码无失真地恢复出原数据。熵编码是建立在随机过程的统计特性基础上的。

设信息源  $X$  可发出的消息符号集合为  $A = \{a_i | i = 1, 2, \dots, m\}$ ，并设  $X$  发出符号  $a_i$  的概率  $p(a_i)$

为，则定义符号  $a_i$  出现的自信息量为：

$$I(a_i) = -\log p(a_i) \quad (6.35)$$

通常，上式中的对数取 2 为底，这时定义的信息量单位为“比特”(bit)。

如果各符号  $a$  的出现是独立的，那么  $X$  发出一符号序列的概率等于各符号的概率之积，因而该序列出现的信息量等于相继出现的各符号的自信息量之和。这类信源称为无记忆信源。

对信息源  $X$  的各符号的自信息量取统计个均，可得平均信息量：

$$H(X) = -\sum_{i=1}^m p(a_i) \log_2 p(a_i) \quad (6.36)$$

称  $H(X)$  为信息源  $X$  的熵 (entropy)，单位为 bit/符号，通常也称为  $X$  的一阶熵，它可以理解为

信息源 X 发任意一个符号的平均信息量。由信息论的基本概念可以知道，一阶熵是无记忆信息源(在无失真编码时)所需数码率的下界。

熵的大小与信源的概率模型有着密切的关系，各个符号出现的概率不同，信源的熵也不同。当信源中各事件是等概率分布时，熵具有极大值。信源的熵与其可能达到的最大值之间的差值反映了该信源所含有的冗余度。信源的冗余度越小，即每个符号所独立携带的信息量越大，那么传送相同的信息量所需要的序列长度越短，符号位越少。因此，数据压缩的一个基本的途径是去除信源的符号之间的相关性，尽可能地使序列成为无记忆的，即前一符号的出现不影响以后任何一个符号出现的概率。

## 6.8.2 CAVLC 的基本原理

在 H.264 的 CAVLC(基于上下文自适应的可变长编码)中，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，取得了极高的压缩比。

CAVLC 用于亮度和色度残差数据的编码。残差经过变换量化后的数据表现出如下特性：4\*4 块数据经过预测、变换、量化后，非零系数主要集中在低频部分，而高频系数大部分是零；量化后的数据经过 zig-zag 扫描，DC 系数附近的非零系数值较大，而高频位置上的非零系数值大部分是+1 和-1；相邻的 4\*4 块的非零系数的数目是相关的。CAVLC 充分利用残差经过整数变换、量化后数据的特性进行压缩，进一步减少数据中的冗余信息，为 H.264 卓越的编码效率奠定了基础。

## 6.8.3 CAVLC 的上下文模型

利用相邻已编码符号所提供的相关性，为所要编码的符号选择合适的上下文模型。利用合适的上下文模型，就可以大大降低符号间的冗余度。在 CAVLC 中上下文模型的选择主要体现在两个方面，非零系数编码所需表格的选择以及拖尾系数后缀长度的更新，以下有详细的论述。

## 6.8.4 CAVLC 的编码过程

### 6.8.4.1 编码非零系数的数目 (TotalCoeffs) 以及拖尾系数的数目(TrailingOnes)

非零系数数目的范围是从 0 到 16，拖尾系数数目的范围是从 0 到 3。如果±1 的个数大于 3 个，只有最后 3 个被视为拖尾系数，其余的被视为普通的非零系数。对非零系数数目和拖尾系数数目的编码是通过查表的方式，共有 4 个变长表格和 1 个定长表格可供选择。其中的定长表格的码字是 6 个比特长，高 4 位表示非零系数的个数 (TotalCoeffs)，最低两位表示拖尾系数的个数(TrailingOnes)。

表格的选择是根据变量 NC (Number Current，当前块值) 的值来选择的，在求变量 NC 值的过程中，体现了基于上下文的思想。除了色度的直流系数外，其它系数类型的 NC 值是根据当前块左边 4\*4 块的非零系数数目 (NA) 和当前块上面 4\*4 块的非零系数数目 (NB) 求得的。当输入的系数是色度的直流系数时，NC= -1。求 NC 的过程见表 6.10，X 表示与当前块同属于一个片并可用。选择非零系数数目和拖尾系数数目的编码表格的过程见表 6.11。CAVLC 的码表见附录。

表 6.10 计算 NC 的值

上面的块(NB)	左边的块(NA)	NC
X	X	(NA+NB)/2
X		NA
	X	NB
		0

表 6.11 选择非零系数数目和拖尾系数数目的编码表格

NC	表格
$0 \leq NC < 2$	变长表格 1
$2 \leq NC < 4$	变长表格 2
$5 \leq NC < 8$	变长表格 3
$NC = -1$	变长表格 4
$NC \geq 8$	定长表格

6.8.4.2 编码每个拖尾系数的符号

对于每个拖尾系数 ( $\pm 1$ ) 只需要指明其符号, 其符号用一个比特表示 (0 表示+, 1 表示-)。编码的顺序是按照反向扫描的顺序, 从高频数据开始。

6.8.4.3 编码除了拖尾系数之外的非零系数的幅值 (Levels)

非零系数的幅值 (Levels) 的组成为两个部分, 前缀 (level\_prefix) 和后缀 (level\_suffix)。levelSuffixsSize 和 suffixLength 是编码过程中需要使用的两个变量。后缀是长度为 LevelSuffixsSize 位的无符号整数。通常情况下变量 levelSuffixsSize 的值等于变量 suffixLength 的值, 有两种情况例外:

- 1. 当前缀等于 14 时, suffixLength 等于 0, levelSuffixsSize 等于 4。
- 2. 当前缀等于 15 时, levelSuffixsSize 等于 12。

变量 suffixLength 是基于上下文模式自适应更新的, suffixLength 的更新与当前的 suffixLength 的值以及已经解码好的非零系数的值 (Level) 有关。suffixLength 数值的初始化以及更新过程如下所示:

- 1. 普通情况下 suffixLength 初始化为 0, 但是当块中有多于 10 个非零系数并且其中拖尾系数的数目少于 3 个, suffixLength 初始化为 1。
- 2. 编码在最高频率位置上的非零系数。
- 3. 如果当前已经解码好的非零系数值大于预先定义好的阈值, 变量 suffixLength 加 1。

决定是否要将变量 suffixLength 的值加一的阈值如表 3 所示。第一个阈值是 0, 表示在第一个非零系数被编码后, suffixLength 的值总是增加 1。

表 6.12 决定增加 suffixLength 的域值

当前 suffixLength	域值
0	0
1	3
2	6
3	12
4	24
5	48
6	N/A

6.8.4.4 编码最后一个非零系数前零的数目(TotalZeros)

TotalZeros 指的是在最后一个非零系数前零的数目, 此非零系数指的是按照正向扫描的最后一个非零系数。例如: 已知一串系数 0 0 5 0 3 0 0 0 1 0 0 -1 0 0 0 0, 最后一个非零系数是-1, TotalZeros

的值等于  $2+3+1+2=8$ 。因为非零系数数目 (TotalCoeffs) 是已知, 这就决定了 TotalZeros 可能的最大值。根据这一特性, CAVLC 在编排 TotalZeros 的码表时做了进一步的优化。

#### 6.8.4.5 编码每个非零系数前零的个数 (RunBefore)

每个非零系数前零的个数 (RunBefore) 是按照反序来进行编码的, 从最高频的非零系数开始。RunBefore 在以下两种情况下是不需要编码的:

1. 最后一个非零系数 (在低频位置上) 前零的个数。
2. 如果没有剩余的零需要编码 ( $\Sigma[\text{RunBefore}] = \text{TotalZeros}$ ) 时, 没有必要再进行 RunBefore 的编码。

在 CAVLC 中, 对每个非零系数前零的个数的编码是依赖于 ZerosLeft 的值, ZerosLeft 表示当前非零系数左边的所有零的个数, ZerosLeft 的初始值等于 TotalZeros, 在每个非零系数的 RunBefore 值编码后进行更新。用这种编码方法, 有助于进一步压缩编码的比特数目。例如: 如果当前 ZerosLeft 等于 1, 就是只剩下一个零没有编码, 下一个非零系数前零的数目只可能是 0 或 1, 编码只需要一个比特。

### 6.8.5 CAVLC 解码过程

CAVLC 解码过程可参考图 6.51 到图 6.53 的程序流程图。CAVLC 解码程序的流程图如图 6.51 所示, 其中解析除了拖尾系数之外的非零系数幅值的子程序流程图如图 6.52 所示, 解析每个非零系数前零的个数 (Run\_Before[i]) 的子程序流程图如图 6.53 所示。

在图 6.51 解码程序流程图中, 第一步根据输入的参数求得输入数据的块类型、输入数据的个数等参数, 这是初始化的工作。随后求变量 NC, 并根据变量 NC 的值来选择所要查的表格, 其中变量 NA 表示与当前块相邻的左边块中非零系数的个数, 变量 NB 表示与当前块相邻的上面块中非零系数的个数。Coeff\_token 是一个句法元素, 以这个句法元素为入口参数, 可查表求得非零系数的个数 (TotalCoeffs)、拖尾系数的个数 (TrailingOnes)。TotalZeros 值的编码方式是查表, 用 TotalCoeff 作为入口参数。

在图 6.52 解析除拖尾系数之外的非零系数幅值的子程序流程图中, 首先根据条件初始化变量 suffixLength 的值, 随后进入循环求解非零系数幅值的过程中。因为拖尾系数已经在前面的过程中解析好了, 此时的非零系数不包括拖尾系数, 所以求解的循环次数是 TotalCoeff-TrailingOnes。在循环中, levelCode 是求解过程中所要使用的中间变量, 判断 levelCode 的奇偶性, 使用不同的公式求解非零系数幅值。循环中最后的步骤是变量 suffixLength 的更新, 如果已经解码好的非零系数的幅值大于相应的预先设定好的域值, 变量 suffixLength 的值加一, 否则 suffixLength 的值不变, 决定 suffixLength 是否需要增加的域值见表 3。

在图 6.53 每个非零系数前零的个数 (Run\_Before[i]) 的子程序流程图中, Run\_before 的值是查表求得的, 表格的入口参数是 Zeroleft, 表示当前非零系数之前 0 的个数。查表解析 Run\_before 的值到 ZeroLeft 等于零或者已经解码到最后一个非零系数 (低频位置) 时结束。

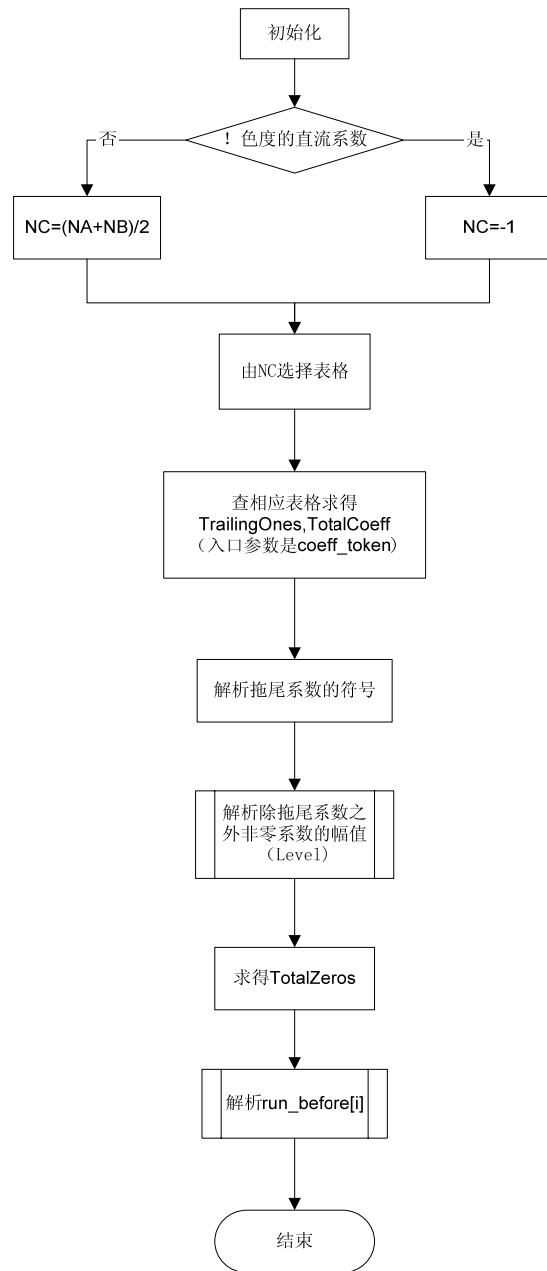


图6.51 解码程序流程图



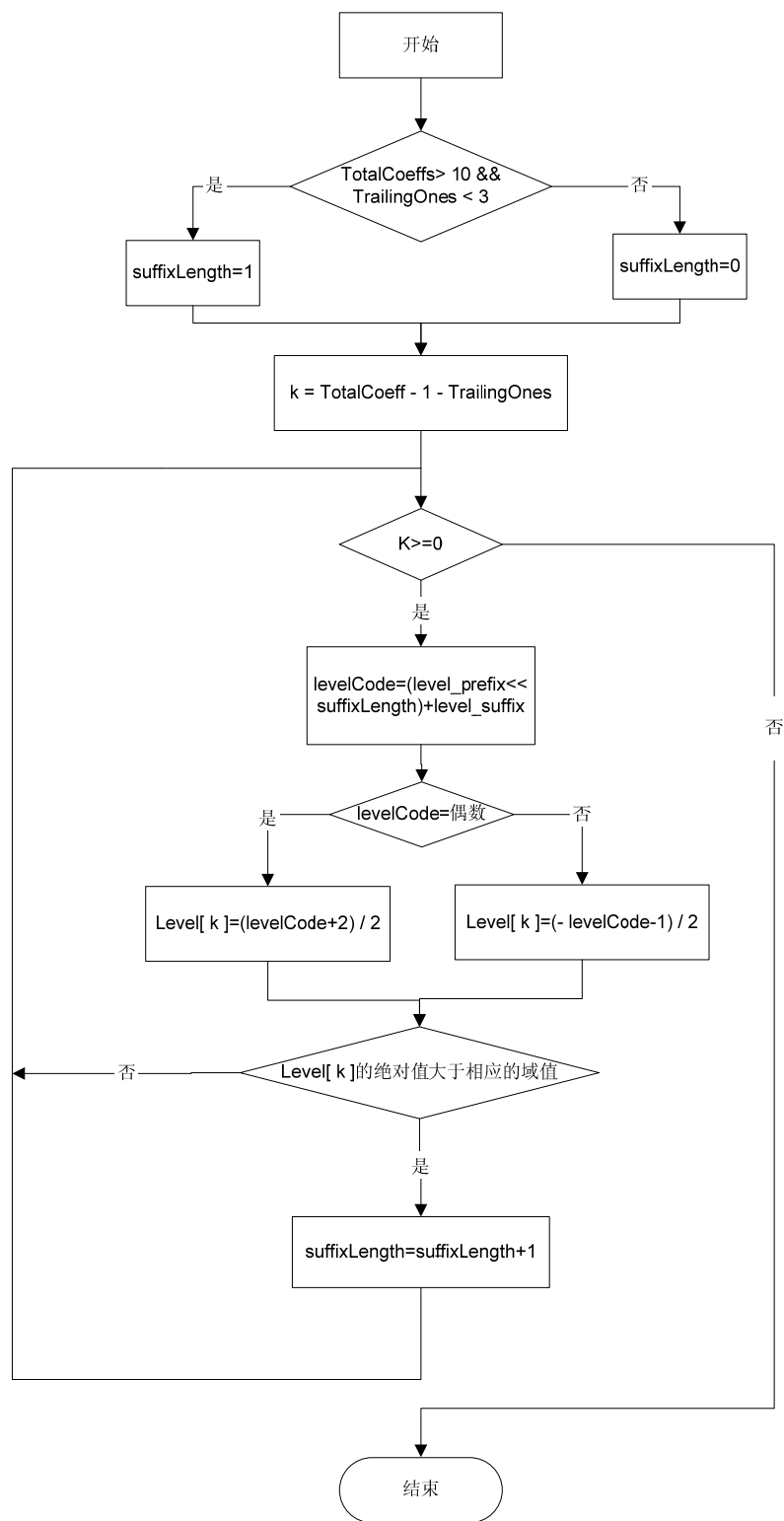


图6.52 解析除拖尾系数之外的非零系数的幅值子程序流程图

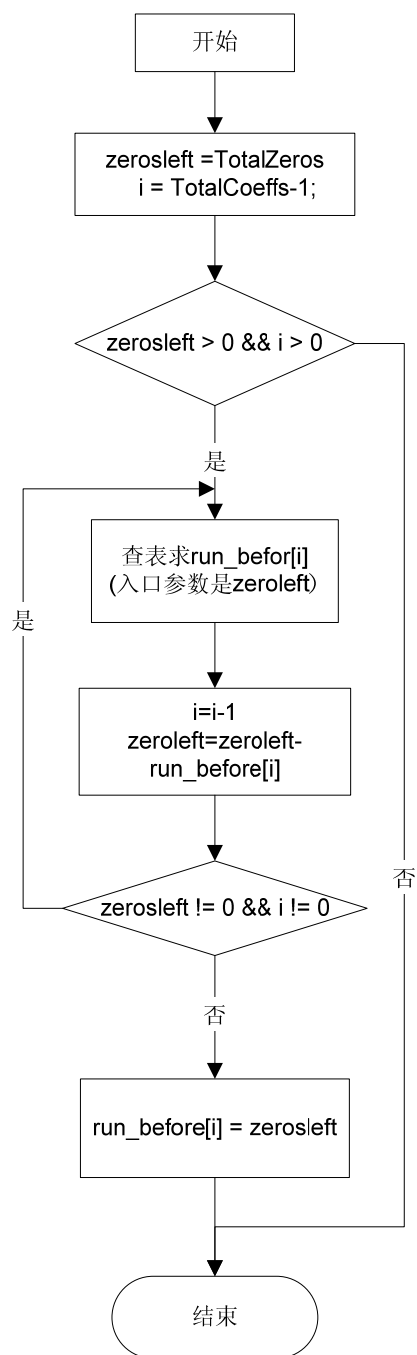


图 6.53 解析每个非零系数前零的数（Run\_Before[i]）的子程序流程图

## 6.8.6 CAVLC 编解码过程实例

4×4 块数据：

0	0	-1	0
5	2	0	0
3	0	0	0
1	0	0	0

数据重排序:

0, 0, 5, 3, 2, -1, 0, 0, 0, 1...

非零系数的数目(TotalCoeffs) =5

以及拖尾系数的数目(TrailingOnes)=2

最后一个非零系数前零的数目(Total\_zeros)=5

变量 NC=3

编码过程:

表 6.13 编码过程

元素	数值	编码
Coeff_token	TotalCoeffs=5,TrailingOnes=2	0000101
Trailing_ones_sign_flag	+	1
Trailing_ones_sign_flag	-	0
Level(1)	2 (suffixLength=0)	001 (前缀)
Level(0)	3 (suffixLength=1)	001 (前缀) 0 (后缀)
Total_zeros	5	101
Run_before(4)	Zreoleft=5,run_before=3	010
Run_before(3)	Zreoleft=2,run_before=0	1
Run_before(2)	Zreoleft=2,run_before=0	1
Run_before(1)	Zreoleft=2,run_before=0	1
Run_before(0)	Zreoleft=2,run_before=2	最后一个系数不需要编码

CAVLC 编码输出的码流: 000010110001001010101011

解码过程:

表 6.14 解码过程

输入码字	元素	数值	输出序列
0000101	Coeff_token	TotalCoeffs=5,TrailingOnes=2	空
1	Trailing_ones_sign_flag	+	1
0	Trailing_ones_sign_flag	-	-1,1
001	Level	2 (suffixLength=0)	2,-1,1
0010	Level	3 (suffixLength=1)	3,2,-1,1
101	Total_zeros	5	5,3,2,-1,1
010	Run_before	3	5,3,2,-1,0,0,0,1
1	Run_before	0	5,3,2,-1,0,0,0,1
1	Run_before	0	5,3,2,-1,0,0,0,1
1	Run_before	0	5,3,2,-1,0,0,0,1

解码后输出序列: 0, 0, 5, 3, 2, -1, 0, 0, 0, 1

## 6.8.7 CAVLC 与 UVLC 比较

CAVLC 代替 H.264 早期草案中提出的 UVLC (统一的可变长编码) 对残差数据进行熵编码。CAVLC 提出的原因可总结为以下的方面:

1. 基于上下文的自适应编码方法可全面的提高编码的质量。
2. 应用 Zig-zag 扫描方法, 将非零系数值 (Level) 和零行程 (Run) 分开编码可提高编码的自适应性, 有助于提高编码的效率。

3. 将非零系数值 (Level) 和零行程 (Run) 分开编码可降低对存储数据的复杂度。

CAVLC 与 UVLC 相比在编码方法上有如下的改进：

1. 可以很容易的观察到，量化后的数据经过 zig-zag 扫描，高频位置上的非零系数值只可能是 +1 或 -1，这些值为 +1、-1 的非零系数被称为拖尾系数 (Trailing Ones)。在 CAVLC 拖尾系数的编码中，只对拖尾系数的数目以及各个系数的符号进行编码。
2. 除了拖尾系数外，其它的非零系数 (Level) 是一维参数，使用的是结构化的 VLC 编码，不需要查表。
3. 在编码零行程 (Run) 时，输入系数的总数目和其中非零系数的数目是已知，这限制了零行程的范围。CAVLC 利用了此特性将零行程分为最后一个非零系数前零的数目 (TotalZeros) 和每个非零系数前零的个数 (RunBefore)，并在码表的安排上进行了进一步的优化。

对 CAVLC 和 UVLC 分别做测试，使用的测试序列是 Silent, 15fps, QCIF；使用的测试软件是 JVT 公布的标准测试软件，测试结果见图 6.54 和图 6.55。可以看出，CAVLC 相对于 UVLC 在编码性能上占有优势。在相同码率的情况下，用 CAVLC 编码的 PSNR 的值高于用 UVLC 编码的 PSNR 的值，并且随着比特率的增加 CAVLC 的优势更加明显。

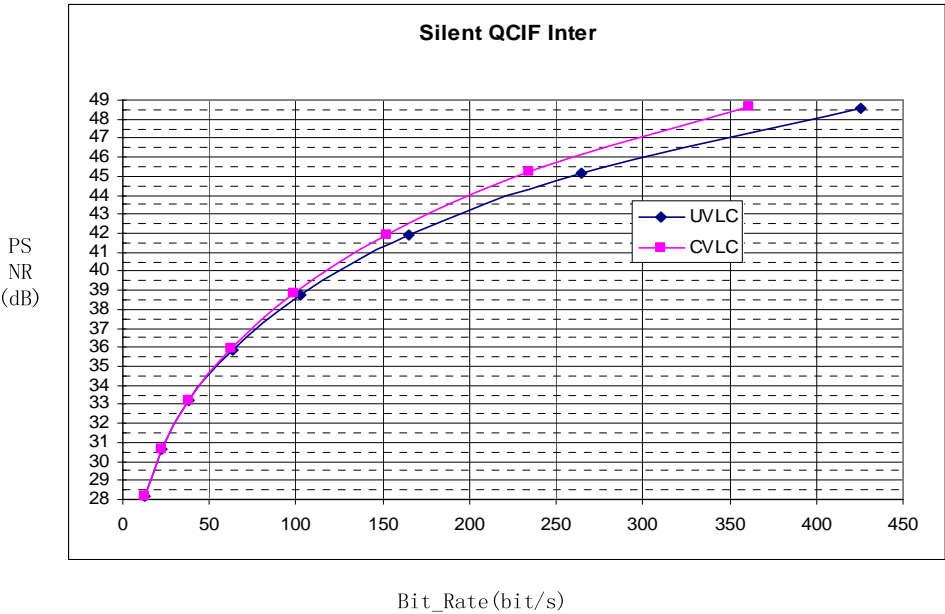


图6.54 实验结果(1)

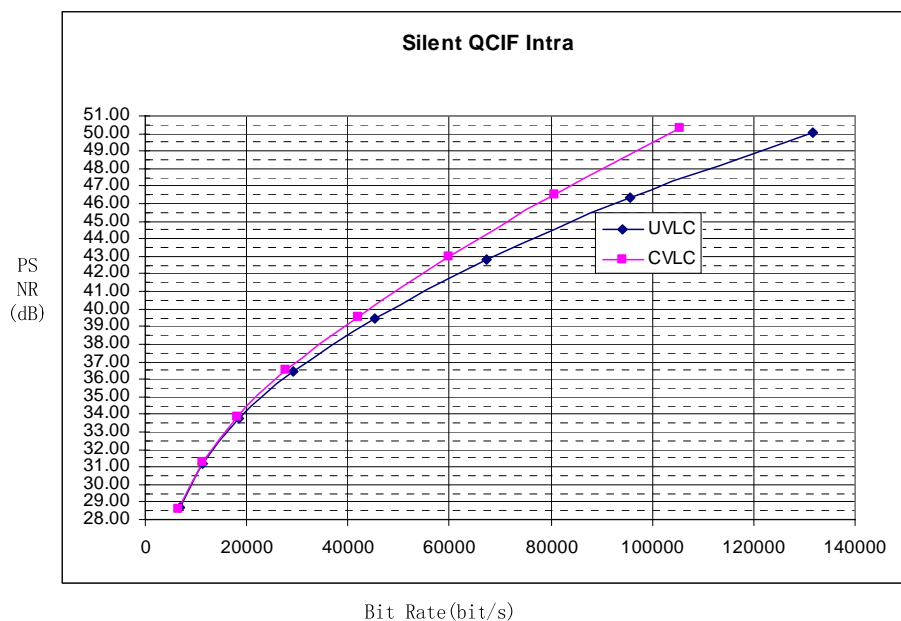


图6.55 实验结果（2）

## 6.9 CABAC(基于上下文的自适应二进制算术熵编码)

### 6.9.1 自适应算术编码

算术编码的思想是用 0 到 1 的区间上的一个数来表示一个字符输入流，它的本质是为整个输入流分配一个码字，而不是给输入流中的每个字符分别指定码字。算术编码是用区间递进的方法来为输入流寻找这个码字的，它从于第一个符号确定的初始区间（0 到 1）开始，逐个字符地读入输入流，在每一个新的字符出现后递归地划分当前区间，划分的根据是各个字符的概率，将当前区间按照各个字符的概率划分成若干子区间，将当前字符对应的子 2 区间取出，作为处理下一个字符时的当前区间。到处理完最后一个字符后，得到了最终区间，在最终区间中任意挑选一个数作为输出。解码器按照和编码相同的方法和步骤工作，不同的是作为逆过程，解码器每划分一个子区间就得到输入流中的一个字符。

#### 6.9.1.1 算法流程

在算术编码的递进计算过程中，编码器必须保存以下变量记录状态：

- 1) 当前区间的下限  $L$
- 2) 当前区间的大小  $R$
- 3) 当前字符  $\text{binval}$
- 4) 各字符的概率  $P_x$

$L$  和  $R$  用来确定当前区间； $P_x$  则是当前区间的划分根据，在二进制编码中，只有 1 和 0 两个字符，所以只需记录  $P_1$  或  $P_0$  即可；最后确定  $\text{binval}$  所在的子区间作为下一个递进中的当前区间。有  $R$  的递进关系：

$$R = R \times P_x \quad (6.37)$$

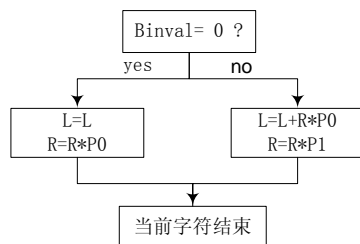


图 6.56 算术编码流程

### 6.9.1.2 自适应

在实际过程中，输入流中字符的概率分布是动态改变的，这需要维护一个概率表去记录概率变化的信息。在作递进计算时，通过对概率表中的值估计当前字符的概率，当前字符处理后，需要重新刷新概率表。这个过程表现为对输入流字符的自适应。编码器和解码器按照同样的方法估计和刷新概率表，从而保证编码后的码流能够顺利解码。

### 6.9.1.3 码流输出

在实际操作过程中，编码器并不是等递进到最终区间才输出码字的，这里有两方面的原因，一是在编码器的递进计算过程中，如果没有输出，信道会出现空闲，形成浪费；二是如果输入流较长时，最终得到的区间非常小，必须以极高的精度来记录  $L$  和  $R$ 。幸运的是，在二进制编码中，区间的上下限以二进制形式表示，每当下限的最高有效位与上限的最高有效位一样时，就可以移出这个比特。这样的方法可以保证编码器在递进计算的同时不断地输出码流。序列出现的可能性越大，区间就越长，确定该区间所需要的比特数就越少。

### 6.9.1.4 算术编码与哈夫曼性能比较

下面通过对比哈夫曼编码来评价算术编码的优势。算术编码的比特率由下式限制：

$$H_N(\mathcal{R})/N \leq R \leq H_N(\mathcal{R})/N + 2/N \quad (6.38)$$

其中， $N$  是编码序列中的符号数而  $H_N(\mathcal{R})$  是序列的  $N$  阶熵。参照矢量哈夫曼编码的情况：

$$H_N(\mathcal{R})/N \leq R \leq H_N(\mathcal{R})/N + 1/N \quad (6.39)$$

可以看到，当  $N$  足够大时，都有：

$$\lim_{N \rightarrow \infty} R_N(\mathcal{R}) = \overline{H}(\mathcal{R}) \quad (6.40)$$

也就是说当输入流足够长时可使比特率接近信源熵率，从这点看，这两者都是优秀的熵编码算法。然而，用哈夫曼编码，必须为所有可能的长度为  $N$  的序列设计和存储码书，这样做的复杂度随  $N$  呈指数增长。用算术编码则不需要预先为每个可能的信源序列指定码书。而是每当所确定区间的下限和上限有公共最高有效位时，就可以连续地得到比特。编码序列的长度可以和信源的长度一样长。因此，实际上，算术编码可以更接近熵率。

算术编码的另一个优点是可以简单地通过更新符号概率表来实现对信源统计特性的自适应。通过对不同上下文用不同的概率表也可以容易地实现条件编码。对于哈夫曼编码，则不得不基于更新的概率表重新设计码书，或对不同的上下文设计多个码表。

由于较高的编码效率和易于自适应，只要所涉及的计算量是能接受的，无疑算术编码比哈夫曼

编码是一种更好的选择。

#### 6.9.1.5 自适应算术编码的计算复杂度及优化

从上文可以看到，算术编码的计算复杂度主要体现在两个方面：

- 1) 概率的估计、更新
- 2) 划分子区间时的乘法运算： $R = R \times Px$

##### 1) CABAC 的概率模型

假设输入流为  $T$ ，当前字符  $\text{binval}$ ，在  $\text{binval}$  之前的字符流为  $z$ ， $z \in T$ ，条件概率  $P(\text{binval}|z)$  就是当前字符的概率估计值。随着条件因子  $z$  的增长，带来的计算量急剧增大，而且每处理一个字符，需要作两次类似的计算，必须找到一种更好的法则来解决这个问题。

首先来研究如果当前字符的概率估计值  $Px$  不是取自  $P(\text{binval}|z)$  会有什么影响。首先毫无疑问这会影响编码效率， $P(\text{binval}|z)$  是取自一个较严格的统计模型，对应于它的输出流的码率能取得对信源熵率的最大逼近；其次，而从解码的可行性上讲，只要保证编码和解码双方在划分当前区间时能得到同样的  $Px$ ，也就是通过同样的法则估计和更新  $Px$ ，就能顺利解码。

**CABAC 在计算的复杂度和编码效率之间作了折中，建立了一个基于查表的概率模型**，将从 0 到 0.5 范围内的概率量化为 64 个值，这些概率对应于 LPS (Least Probability Symbol) 字符，则 MPS (Most Probability Symbol) 字符的概率为  $1 - P_{lps}$ 。字符的概率估计值被限制在表内，概率的刷新也不是去计算  $P(\text{binval}|z)$ ，而是按照某种法则在表中查找。

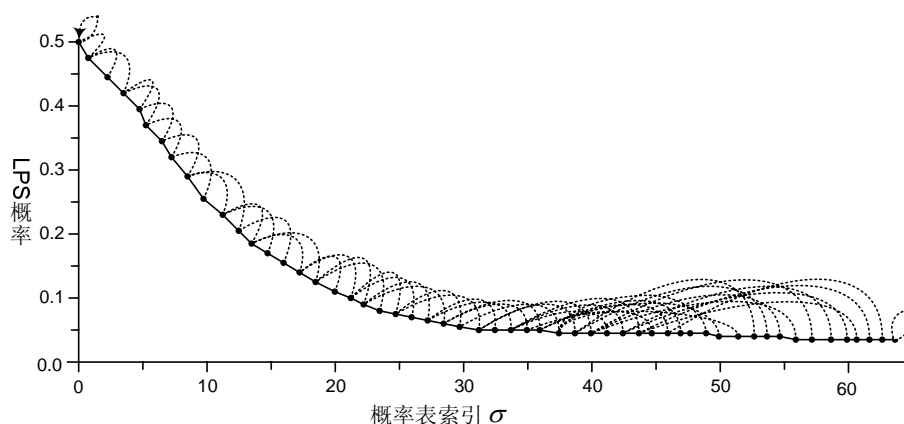


图 6.57 CABAC 概率估计与刷新模型

如图 6.57 所示，列出了 LPS 被量化后的概率值，这些值以  $\sigma$  为编号，实线和虚线指示了概率的刷新值。在处理当前字符  $\text{binval}$  时，概率的刷新有两个方向：一，如果当前字符是 LPS，则  $P_{lps}$  变大，在图上顺着虚线往左寻找；二，如果当前字符是 MPS，则  $P_{lps}$  变小，在图上顺着实线往右寻找。可以看到，在 CABAC 建立的这个概率模型中，出现 MPS 时的刷新值都只是简单地指向当前值的下一位，即  $\sigma \rightarrow \sigma + 1$ ，这一点可以被利用来降低计算量。

在 CABAC 建立的概率模型中，有三个值是较特殊的： $\sigma = 0$  时，LPS 的概率已经达到了最大值 0.5，如果下一个出现的字符仍是 LPS，则此时 LPS 和 MPS 的字符交换位置； $\sigma = 63$  对应着 LPS 的最小概率值，但它并没有被纳入 CABAC 的概率估计和更新的范围，这个值被用作特殊的场合，传达特殊的信息。比如，当解码器检测到当前区间的划分依据是这个概率值时，认为这表示当前流的结束； $\sigma = 62$ ，这是表中可用的最小值，它对应的刷新值是它自身，当 MPS 连续出现，LPS 的

概率持续减小，直到  $\sigma = 62$ ，保持不变。

## 2) 乘法优化

CABAC 的概率模型很有效地降低了了概率估计和刷新中的计算量，而对于在算术编码频繁使用的乘法运算  $R = R \times Px$ ，CABAC 也应用了类似的思想。

CABAC 首先建立一个  $4 \times 64$  的二维表格，存储预先计算好的乘法结果。表格的入口参数毫无疑问一个来自  $Px$ ，另一个来自  $R$ 。 $Px$  可以直接以  $\sigma$  作为参数，下面的式子给出  $R$  的量化方法：

$$\rho = (R \gg 6) \& 3 \quad (6.41)$$

每次在需要做乘法运算时，携带  $\rho$  和  $\sigma$  进行查表操作就得到结果。

建立了概率模型和乘法模型后，在递进计算过程中 CABAC 必须保存以下变量记录状态：

- 1) 当前区间的下限  $L$
- 2) 当前区间的大小  $R$
- 3) 当前 MPS 字符  $\varpi$
- 4) LPS 的概率编号  $\sigma$

算法流程：

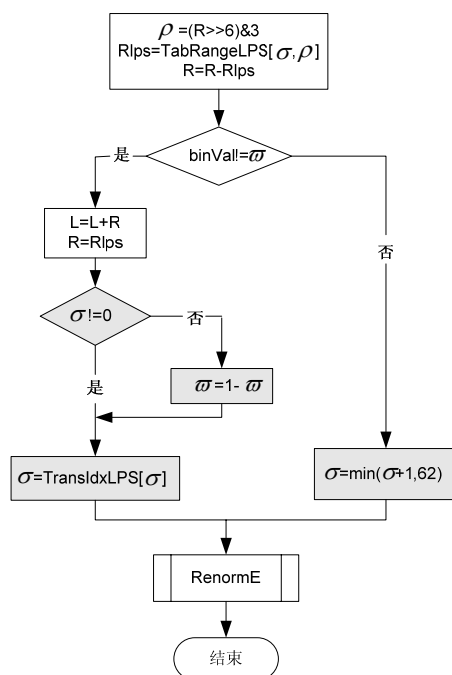


图 6.58 CABAC 做算术编码时流程图

图中，灰色部分是概率的刷新部分。表 TabRangeLPS 存储预先计算好的乘法结果，表 TransIdxLPS 是图 2 对应的概率表。

## 6.9.2 上下文模型

### 6.9.2.1 算术编码的生命期

算术编码是对整个流分配码字，但考虑到如果有某个比特丢失，编码和解码将错位。为了将差错控制在一定范围内，CABAC 将片（slice）作为算术编码的生命期，在每个片开始时，CABAC 进行初始化，按照一定的法则为编码器指定初始  $\varpi$ 、 $\sigma$ ，并初始化  $[0,1)$  为当前区间。



### 6.9.2.2 CABAC 的上下文模型

即使对片内的数据，CABAC 也不是将它们作为整体来处理的，而是继续分割为若干个子部分，分别编码。除了上文提到的差错控制的原因外，也是由于如果输入流过长，则要求 L 和 R 必须有足够的精度和长度来保存中间数据，这对于编码器是个不小的负担。

**H.264 将一个片内可能出现的数据划分为 399 个上下文模型,每个模型以 ctxIdx 标识，在每个模型内部进行概率的查找和更新。**H.264 共要建立 399 个概率表，每个上下文模型都独立地使用对应的表维护概率状态。这些模型的划分精确到比特，几乎大多数的比特和它们邻近的比特处于不同的上下文模型中。

解码器对于输入的每一个比特首先要做的工作是查找它属于哪个上下文模型，然后查找该上下文模型对应的概率表以递进区间。查找某个比特对应的上下文模型一般有以下两个步骤：

1. 确定该比特所属的句法元素，H.264 对每个句法元素都分配了一个上下文模型的区间，该句法元素中的每个比特的上下文模型的 ctxIdx 都在这一区间。表 6.15 描述了用 CABAC 编码的各句法元素所在上下文模型区间的起始 ctxIdx。句法元素的具体含义在第七章句法与语义中详细介绍。

表 6.15 句法元素与上下文模型区间

句法元素	片类型			
	SI	I	P, SP	B
mb_skip_flag			11-13	24-26
mb_field_decoding_flag	70-72	70-72	70-72	70-72
mb_type	0-10	3-10	14-20	27-35
coded_block_pattern (luma)	73-76	73-76	73-76	73-76
coded_block_pattern (chroma)	77-84	77-84	77-84	77-84
mb_qp_delta	60-63	60-63	60-63	60-63
prev_intra4x4_pred_mode_flag	68	68	68	68
rem_intra4x4_pred_mode	69	69	69	69
intra_chroma_pred_mode	64-67	64-67	64-67	64-67
ref_idx_l0			54-59	54-59
ref_idx_l1				54-59
mvd_l0[ ][ ][ 0 ]			40-46	40-46
mvd_l1[ ][ ][ 0 ]				40-46
mvd_l0[ ][ ][ 1 ]			47-53	47-53
mvd_l1[ ][ ][ 1 ]				47-53
sub_mb_type			21-23	36-39
coded_block_flag	85-104	85-104	85-104	85-104
significant_coeff_flag[ ]	105-165, 277-337	105-165, 277-337	105-165, 277-337	105-165, 277-337
last_significant_coeff_flag[ ]	166-226, 338-398	166-226, 338-398	166-226, 338-398	166-226, 338-398

coeff_abs_level_minus1[ ]	227-275	227-275	227-275	227-275
---------------------------	---------	---------	---------	---------

2. 按照某个法则为当前比特在 a)中得到的区间中找到所对应的 **ctxIdx**。该法则对于每个句法元素都不同。这个法则一般是用表来定义的，表 2 描述了大多数句法元素为所属各比特查找 **ctxIdx** 的法则。在这个表中，**binIdx** 是各比特在对应句法元素中的序号。**ctsOffset** 是表 6.15 中所示各句法元素对应上下文模式区间的起始偏移。我们看到，在表 6.16 中某些比特对应多个上下文模型，这一般是在解码中需要根据前后宏块在空间上的相关性再作进一步确定。

表 6.16 句法元素中各比特的 **ctxIdx**

<b>ctxIdxOffset</b>	<b>binIdx</b>						
	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>&gt;= 6</b>
0	0,1,2	无	无	无	无	无	无
3	0,1,2	ctxIdx=276	3	4	5,6	6,7	7
11	0,1,2	无	无	无	无	无	无
14	0	1	2,3	无	无	无	无
17	0	ctxIdx=276	1	2	2,3	3	3
21	0	1	2	无	无	无	无
24	0,1,2	无	无	无	无	无	无
27	0,1,2	3	4,5	5	5	5	5
32	0	ctxIdx=276	1	2	2,3	3	3
36	0	1	2,3	3	3	3	无
40	0,1,2	3	4	5	6	6	6
47	0,1,2	3	4	5	6	6	6
54	0,1,2,3	4	5	5	5	5	5
60	0,1	2	3	3	3	3	3
64	0,1,2	3	3	无	无	无	无
68	0	无	无	无	无	无	无
69	0	0	0	无	无	无	无
70	0,1,2	无	无	无	无	无	无
73	0,1,2,3	0,1,2,3	0,1,2,3	0,1,2,3	无	无	无
77	0,1,2,3	4,5,6,7	无	无	无	无	无
276	0	无	无	无	无	无	无

### 6.9.3 对输入流预编码

CABAC 围绕算术编码的特性作了许多优化，这其中也包括从统计角度对输入流作的一套预编码方法。由图 3 可以看到，当前处理的字符为 MPS 时，区间递进只是子区间的长度发生改变，而作为影响实际输出值的 L 却并没有变化。这个现象意味着如果输入流中连续出现大量 MPS，或者 MPS 对 LPS 的概率比非常高时，可以达到极高的压缩效果。算术编码对这种输入流的压缩性能达到最优，编码输出的码率也更能接近信源熵率。由此 CABAC 体系包含了一个预编码过程，将输入流重新编码后再进行算术编码。这个预编码过程叫做输入流的二进制化，经它编码输出的是 MPS 概率极高的比特流。

CABAC 中对不同的句法元素一共应用了四种二进制化方法：

- U

- TU
- UEGK
- FL

表 6.17 是其中 U 变换的码表其它二进制化编码方法类似。binIdx 是编码后各比特的序号。

表 6.17 U 二进制化码表

编码前	编码后比特流					
0	0					
1	1	0				
2	1	1	0			
3	1	1	1	0		
4	1	1	1	1	0	
5	1	1	1	1	1	0
...						
binIdx	0	1	2	3	4	5

## 6.9.4 初始化

前文提到，CABAC 的生命期是片，每个片开始时，要对 399 种上下文模型全部进行初始化工作。初始化的步骤是，

- 将递进区间复位到[0,1)
- 为每个上下文模型指定一个初始的  $\varpi$ 、 $\sigma$ 。这又由以下几步得到：
  - H.264 为每个上下文模型定义了初始化常量 m、n，通过查表获得上下文模型相对应的 m、n。
  - 按照如下算法计算  $\varpi$ 、 $\sigma$ ：

```

preCtxState = Clip3( 1, 126, ( ( m * SliceQPY ) >> 4 ) + n )

if( preCtxState <= 63 ) {
     $\sigma$  = 63 - preCtxState
     $\varpi$  = 0
} else {
     $\sigma$  = preCtxState - 64
     $\varpi$  = 1
}

```

式中，函数 Clip3(a,b,c)表示将 c 的值限制在[a,b]。preCtxState 是一个中间变量。

## 6.9.5 结论

**CABAC 中内建了由大量实验统计而得到的概率模型。**在编码过程中，CABAC 根据当前所要编码的内容以及先前已编码好内容，动态地选择概率模型来进行编码，并实时更新相对应的概率模型。并且，CABAC 在计算量和编码速度上进行了优化，用了量化查表、移位、逻辑运算等方法代替复杂的概率估计和乘法运算。在实际应用中，CABAC 与其它主流的熵编码方式相比有更高的编码效率，用一组质量在 28~40dB 的视频图像做测试，应用 CABAC 可使比特率进一步提高 9%~14%。图 6.59 描述了图象质量在各信噪比时 CABAC 节省码率的性能。

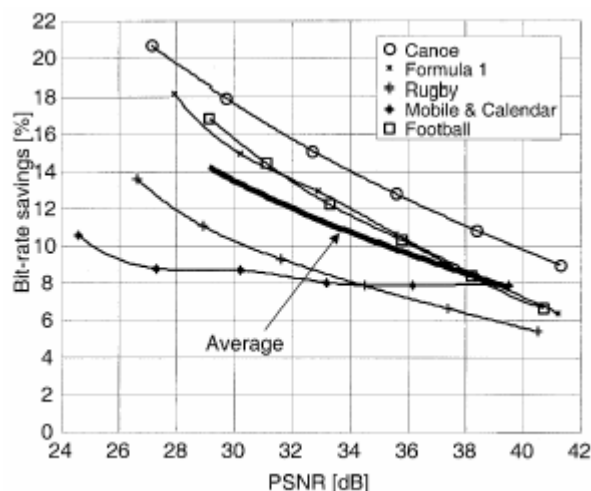


图 6.59 各测试序列中 CABAC 节省码率的性能

## 6.10 码率控制

在 H.264 视频编码标准中仅仅规定了编码后比特流的句法结构和解码器的结构，而对于编码器的结构和实现模式没有具体的规定。然而无论编码器的结构如何，相应的视频编码的控制都是编码器实现的核心问题。在对数字视频信号进行压缩编码时，编码器通过相应的编码控制算法以确定各种编码模式，如宏块的划分类型、运动矢量以及量化参数等，已选定的各种编码模式进一步确定了编码器输出比特流的比特率和失真度。

H.264 编码器采用了基于 Lagrangian 优化算法的编码控制模型，其编码性能相较于以往的所有编码标准有了重大提高。

### 6.10.1 基于 Lagrangian 优化算法的 H.264 编码控制模型

视频编码的控制都是编码器实现的核心问题。由于视频序列中的图像内容随着空间与时间的不同而变化很大，需要为图像的不同部分选择不同的编码参数进行压缩编码，而编码控制的目的就是确定一组编码参数。H.264 编码器采用基于 Lagrangian 优化算法的率失真优化模型实现视频编码的控制，其实现简单而且效率高。下面将分析 Lagrangian 优化算法及其在 H.264 视频编码控制中的应用。

#### 6.10.1.1 Lagrangian 优化算法

考虑K个信源样本值的集合 $S=(S_1, \dots, S_K)$ ，其中 $S_K$ 可以是矢量或标量。每一个样本值 $S_K$ 可以通过选取编码模式集 $O_k=(O_{k1}, \dots, O_{kN})$ 中的某些编码模式 $I_k (I_k \in O_k)$ 进行压缩编码。因此对应于样本值集合 $S$ 存在相应的编码模式集合 $I=(I_1, \dots, I_K)$ 。在给定的限定码率 $R_c$ 下，对于给定信源样本序列所选的编码模式，应使编码后的失真度最小，如（1）式所示。

$$\min_I D(S, I) \quad R(S, I) \leq R_c \quad (6.42)$$

（6.42）式中， $D(S, I)$ 与 $R(S, I)$ 分别表示输出比特流的失真度和码率，其中比特流由采用编码模式 $I$ 对样本 $S$ 进行编码并进行量化后输出。

在实际应用中，通常采用下式来选取编码模式。

$$I^* = \arg \min J(S, I | \lambda) \quad \text{其中 } J(S, I | \lambda) = D(S, I) + \lambda \times R(S, I) \quad (6.43)$$

(6.43) 式中 $\lambda$ 是Lagrange参数。对于样本 $S$ 及其选定的编码模式 $I$ ，当其编码后得到的比特率和失真度的线性组合 $J(S, I | \lambda)$  (Lagrangian代价函数) 最小时，此时的编码模式是最优的。

考虑某一样本 $S_k$ ，可认为其编码后的比特率和失真度仅与相应的编码模式 $I_k$ 有关，因此有下面2式成立。

$$J(S_k, I | \lambda) = J(S_k, I_k | \lambda) \quad (6.44)$$

$$\min_I \sum_{k=1}^K J(S_k, I | \lambda) = \sum_{k=1}^K \min_{I_k} J(S_k, I_k | \lambda) \quad (6.45)$$

因此，只要分别对每一个样本 $S_k \in S$ 选择最优的编码模式，便可以很容易的得到 $J(S, I | \lambda)$ 的最小值，从而实现相应的编码控制。

#### 6.10.1.2 编码控制模型

由于编码后比特流的比特率和失真度与时间和空间的关系密切，基于 Lagrangian 优化算法的编码控制不可能在混合视频编码器中简单的实现。假设图像序列  $s$  被分割为  $K$  个不同的块  $A_k$ ，相应的像素用  $S_k$  表示。编码  $S_k$  所选择的编码模式  $O_k$  分为帧内模式和帧间模式两类。每种模式均包括预测编码的模式以及相应的编码参数，其中编码参数为变换系数和量化参数等，对于帧间模式编码参数还应包括一个或多个运动矢量。在对图像序列  $s$  进行基于块的混合视频编码时，对于每块  $S_k$  所选定的编码模式应当使编码后的 Lagrangian 代价函数  $J(S, I | \lambda)$  达到最小，当且仅当此时认为基于块的混合视频编码器达到最优化。

对帧间模式的选择，通常通过搜索使得编码后 Lagrangian 代价函数  $J(S, I | \lambda)$  最小化的运动矢量实现，相应的运动矢量作为编码参数被编码并传输。因此在编码控制模型中，宏块分割模式的判决与帧间模式运动估计的最佳比特分配这两个问题将会被分别处理。

在 Lagrange 参数  $\lambda_{MODE}$  与量化参数  $Q$  选定后，H.264 的编码器通过最小化 Lagrangian 代价函数实现对每一个宏块的编码模式的选定。宏块  $S_k$  的 Lagrangian 代价函数如 (6.46) 式所示。

$$J_{MODE}(S_k, I_k | Q, \lambda_{MODE}) = D_{REC}(S_k, I_k | Q) + \lambda_{MODE} \times R_{REC}(S_k, I_k | Q) \quad (6.46)$$

$I_k$  为相应宏块的编码模式。

在不同编码模式下，编码后比特流的比特率  $R_{REC}$  与失真度  $D_{REC}$  的计算并不完全相同。在帧内模式下， $R_{REC}(S_k, INTRA | Q)$  为熵编码后比特流的比特率，失真度  $D_{REC}(S_k, INTRA | Q)$  则由宏块的原始像素和重建像素决定，且共有 2 种计算方式，分别如式 (6.47)、(6.48) 所示。

$$SSD = \sum_{(x,y) \in A} |s[x, y, t] - s'[x, y, t]|^2 \quad (6.47)$$

$$SAD = \sum_{(x,y) \in A} |s[x, y, t] - s'[x, y, t]| \quad (6.48)$$

其中  $A$  为当前的宏块。

对于 SKIP 模式，由于无需残差信号，因此比特率  $R_{REC}(S_k, INTRA | Q)$  与失真度  $D_{REC}(S_k, INTRA | Q)$  与量化参数无关。其中失真度  $D_{REC}(S_k, INTRA | Q)$  由宏块的原始像素值和预测像素值决定，而比特率  $R_{REC}(S_k, INTRA | Q)$  则在 H.264 中被近似为 1bit / MB。

在帧间模式下由于采用了基于块的运动估计，Lagrangian 代价函数的计算比较于帧内模式与 SKIP 模式要复杂。对于采用帧间编码模式的  $A \times B$  大小的块  $S_i$ ，在给定的 Lagrange 参数  $\lambda_{MOTION}$  和参考图像  $s'$  的情况下，通过最小化 Lagrangian 代价函数来实现块  $S_i$  的运动估计，如式 (6.49) 所示。

$$m_i = \arg \min_{m \in M} \{D_{DFD}(S_i, m) + \lambda_{MOTION} R_{MOTION}(S_i, m)\} \quad (6.49)$$

其中 M 为可能的编码模式的集合， $R_{DFD}(S_i, m)$  为传输运动矢量  $(m_x, m_y, m_t)$  所需的比特数，失真度  $D_{DFD}$  由式 (6.50) 或式 (6.51) 得到。

$$SSD = \sum_{(x,y) \in A_i} |s[x, y, t] - s'[x - m_x, y - m_y, t - m_t]|^2 \quad (6.50)$$

$$SAD = \sum_{(x,y) \in A_i} |s[x, y, t] - s'[x - m_x, y - m_y, t - m_t]| \quad (6.51)$$

在运动估计时水平与垂直方向的搜索范围为  $\pm 32$  个整像素，并采用一帧或多帧参考图像。

为寻找满足式 (6.49) 要求的运动矢量  $m_t$ ，首先在整像素位置进行运动估计的运算，求得满足式 (6.49) 要求的运动矢量后，需进一步确定周围半像素位置的运动矢量是否可使 Lagrangian 代价函数进一步降低。由于在 H.264 中采用了  $1/4$  像素的运动估计精度，之前确定的半像素周围  $1/4$  像素位置的运动矢量被进一步考察，以确定当采用此  $1/4$  像素精度的运动矢量后 Lagrangian 代价函数是否获得进一步的降低。通过以上分析可知，最终选定使得 Lagrangian 代价函数最小的运动矢量具有  $1/4$  像素精度。

同帧内模式相同，在帧间模式下比特率  $R_{REC}(S_k, INTER|Q)$  为熵编码后比特流的比特率，失真度  $D_{REC}(S_k, INTER|Q)$  则由宏块的原始像素和重建像素决定，由式 (6.46) 或式 (6.47) 得到。

在 H.264 视频编码控制模型中， $\lambda_{MODE}$  由量化参数确定，如式 (6.52) 所示。

$$\lambda_{MODE} = 0.85 \times 2^{(Q-12)/3} \quad (6.52)$$

另一个 Lagrange 参数  $\lambda_{MOTION}$  与  $\lambda_{MODE}$  有关，由式 (6.53) 或式 (6.54) 确定。其中式 (6.53) 对应于式 (6.47) 与式 (6.50)，式 (13) 对应于式 (7) 与式 (10)。

$$\lambda_{MOTION} = \lambda_{MODE} \quad (6.53)$$

$$\lambda_{MOTION} = \sqrt{\lambda_{MODE}} \quad (6.54)$$

在 H.264 中，通常通过速率控制相关算法选择合适的量化参数，并通过相应的 Lagrange 参数进行视频编码控制。

由上所述，H.264 视频编码器中的基于 Lagrangian 优化算法的编码控制模型可由图 6.60 表示。

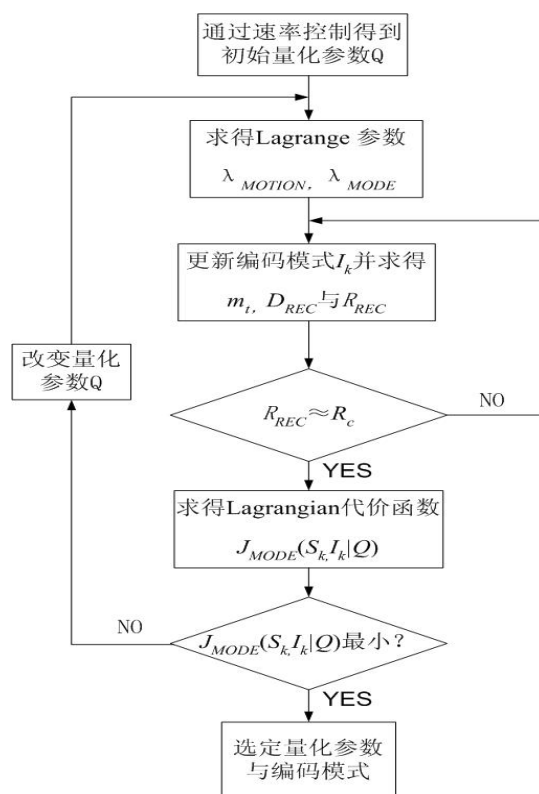


图 6.60 H.264 视频编码控制模型

## 6.10.2 实验结果和性能分析

分别针对 H.264 在流媒体与视频会议中的应用进行实验。视频会议与流媒体应用的主要区别在于，视频会议需要满足满足低时延和实时的要求，而流媒体的相关应用则侧重于图像细节的处理。因此基于 H.264 编码标准的视频会议与流媒体应用所实现的编码器的构成区别很大。在合理的为两种编码器选择编码工具的基础上，实现了基于 Lagrangian 优化算法的编码控制模型，并与其他编码标准 MPEG-2, MPEG-4, H.263 的编码器的编码性能进行比较。

### 6.10.2.1 流媒体应用中的实验结果和性能分析

针对于流媒体应用，基于 H.264 测试模型 JM-61e 所实现的主档次（MP）编码器的构成特性如表 6.18 所示，并在此基础上实现了基于 Lagrangian 优化算法的编码控制模型。考虑到流媒体应用的特点，选取 4:2:0 CIF 格式标准测试序列 Mobile&Calendar 进行测试，输入帧频为 30f / s，长度为 8.33 秒。该视频序列具有复杂的运动和较高的空间色彩细节，适于测试流媒体应用中编码器的性能。

表 6.18 编码器的构成特性

运动估计中整像素搜索范围	±32
熵编码方式	CABAC
参考帧数目	5
每一组图像序列的结构	IPBBP

在实验中，设定编码后帧率为 30f / s，实验结果如图 6.61 所示。其中在编码码率分别为 512Kbits / s，1024 Kbits / s 的条件下，得到相应输出比特率的码率和亮度色度分量的信噪比如表 6.19 所示。为公平比较编码性能，在同样条件下对基于 H.263 HLP, MPEG2, MPEG4 ASP 编码标准的编码器也进行了测试，相应的测试模型分别为 TMN-10, TM-5, VM-18。

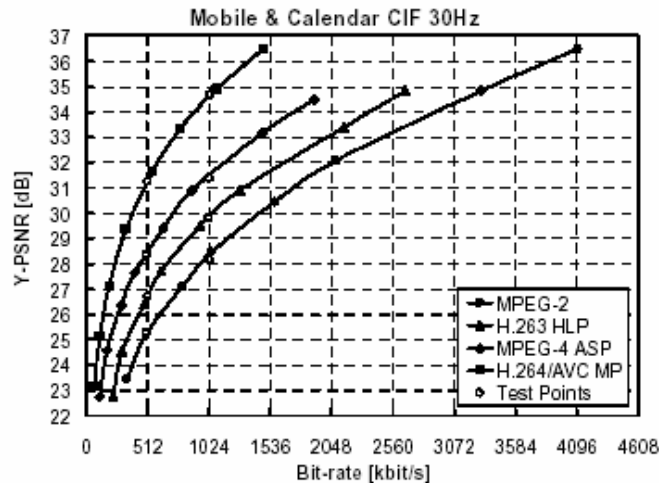


图 6.61 流媒体应用中各编码器的性能比较

表 6.19 流媒体应用中各编码器的实验结果

序列编码码率	编码标准	输出码率 (Kbits / s)	亮度分量 (Y) 信噪比 (dB)	色度分量 (U) 信噪比 (dB)	色度分量 (V) 信噪比 (dB)
512Kbits / s	MPEG2	506.26	25.31	30.26	30.47
	H.263 HLP	513.05	26.74	32.40	32.85
	MPEG4 ASP	505.03	28.36	33.12	33.54
	H.264 MP	512.58	31.27	35.18	35.65
1024 Kbits / s	MPEG2	1029.58	28.16	33.00	33.27
	H.263 HLP	1024.27	29.82	34.43	34.83
	MPEG4 ASP	1029.18	31.37	35.29	35.74
	H.264 MP	1026.00	34.64	37.27	37.74

通过表 6.19 可知，实现了基于 Lagrangian 优化算法的编码控制模型的 H.264 编码器，其编码性能也即输出亮度色度分量的信噪比相较于 MPEG4 ASP 提高了 1-3dB，相较于 H.263HLP 提高了 1-5dB，相较于 MPEG2 提高了 3-6dB。

#### 6.10.2.2 视频会议应用中的实验结果和性能分析

针对于视频会议应用，基于 H.264 测试模型 JM-61e 所实现的基本档次（Baseline）编码器的构成特性如表 6.20 所示，并在此基础上实现了基于 Lagrangian 优化算法的编码控制模型。考虑到视频会议应用的特点，选取 4:2:0 CIF 格式标准测试序列 Paris 进行测试，输入帧频为 30f / s，长度为 10 秒。该视频序列中的物体几乎处于静止状态，适于测试视频会议应用中编码器的性能。

表 6.20 编码器的构成特性

运动估计中整像素搜索范围	±32
熵编码方式	CAVLC
参考帧数目	5
每一组图像序列的结构	IPPPP

在实验中，设定编码后帧率为 15f / s，实验结果如图 3 所示。在编码码率为 128Kbits / s 的条件下，得到相应输出比特率的码率和亮度色度分量的信噪比。为公平比较编码性能，在同样条件下对基于 H.263 Baseline, H.263 CHC, MPEG4 SP 编码标准的编码器也进行了测试，相应的测试模型分



别为 TMN-10, TMN-10, VM-18。

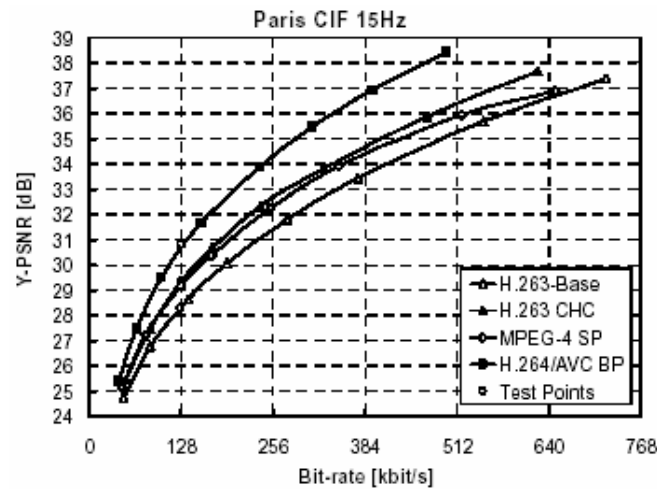


图 6.62 视频会议应用中各编码器的性能比较

表 6.21 视频会议应用中各编码器的实验结果

序列编码码率	编码标准	输出码率 (Kbits / s)	亮度分量 (Y) 信噪比 (dB)	色度分量 (U) 信噪比 (dB)	色度分量 (V) 信噪比 (dB)
128Kbits / s	H.263Baseline	127.38	28.30	33.30	33.84
	H.263 CHC	128.29	29.34	35.56	36.32
	MPEG4 SP	127.95	29.18	33.59	34.25
	H.264Baseline	128.52	30.81	35.80	36.18

通过表 6.21 可知，实现了基于 Lagrangian 优化算法的编码控制模型的 H.264 编码器，其编码性能也即输出亮度色度分量的信噪比相较于 MPEG4 提高了 1-2dB，相较于 H.263 Baseline 提高了 1-3dB。

以上两个实验结果证实：实现了基于 Lagrangian 优化算法的编码控制模型的 H.264 编码器，其编码性能相较于以往的所有编码标准有了重大提高，并在流媒体及视频会议等视频应用中显示出了巨大潜力。

6.11 去方块滤波

H.264/MPEG-4 AVC 视频编码标准中，在编解码器反变换量化后图像会出现方块效应。其产生的原因有两个。最重要的一个原因是基于块的帧内和帧间预测残差的 DCT 变换。变换系数的量化过程相对粗糙，因而反量化过程恢复的变换系数带有误差，会造成在图像块边界上的视觉不连续。第二个原因来自于运动补偿预测。运动补偿块可能是从不是同一帧的不同位置上的内插样点数据复制而来。因为运动补偿块的匹配不可能是绝对准确的，所以就会在复制块的边界上产生数据不连续。当然，参考帧中存在的边界不连续也被复制到需要补偿的图像块内。尽管 H.264/MPEG-4 AVC 采用较小的 4x4 变换尺寸可以降低这种不连续现象，但仍需要一个去方块滤波器以最大程度提高编码性能。

在视频编解码器中加入去方块滤波器的方法有两种：**后置滤波器**和**环路滤波器**。后置滤波器只处理编码环路外的显示缓冲器中的数据，所以它不是标准化过程中的规范内容，在标准中只是可选项。相反，**环路滤波器处理编码环路中的数据。在编码器中，被滤波的图像帧作为后续编码帧运动补偿的参考帧；在解码器中，滤波后的图像输出显示。**这要求所有与本标准一致的解码器采用同一个滤波器以与编码器同步。当然如果有必要，解码器也还可以在使用环路滤波器的同时使用后置滤波器。

在编码环路中使用滤波器比后置滤波有几点优点。首先，环路滤波器可以保证不同水平的图像质量。其次，在解码器端没有必要再为滤波器准备额外帧缓存。第三，经验试验已显示环路滤波比后置滤波更能增加视频流的主客观质量，同时有效降低解码器的复杂度。

尽管有以上这些优点，环路滤波器的复杂度还是较高的。即使经过很大努力进行滤波算法的时间优化，去除其中的乘除法，**滤波器也轻易地达到解码器计算复杂度的三分之一。**

**高复杂度的主要原因是滤波器的高度自适应性**，它需要对方块边界及样点量化值进行条件判断和处理。这样，在算法的主要内部循环中不可避免地出现条件分支。众所周知，这是很耗时的。另一个高复杂度的原因是 H.264 中编码算法中残差编码的尺寸。对于 4×4 的方块大小及平均在每个方向上滤波 2 个点，这样几乎图像中每个点都要被调入到内存中，要么被修正，要么用来判断边界点是否要被修正。而对 H.263 环路滤波器或任何 MPEG-4/H.263 后置滤波器来说不是这样的，因为它们对 8×8 方块进行操作。

下面介绍的自适应去方块滤波器利用简单的算法可靠地提高图像的主客观质量。其较好的性能是因为可靠地区分了真实的和人为的图像边界，并有效地滤除后者。在相同的 PSNR 下可以节省码流超过 9%，并同时明显地提高图像视觉质量。因为去方块滤波在 H.265 编码器和解码器中的作法是一致的，所以在第八章解码器中就不再赘述。

### 6.11.1 去方块滤波基本概念

图 6.63(a)显示不采用去方块滤波器的编解码器的效果图。基于上述原因，图中在 DCT 变换边界上有明显的痕迹，呈现出方块形状。图 6.63(b)则是采用去方块滤波器的编解码器对同一幅图像的处理效果图。图(a)中的方块已不明显了。

显然，去方块滤波器的作用是去除 H.264 编解码算法带来的方块效应。但是，如果在 DCT 边界上，正好是图像的边界，如家具边等，若不加以判断而误认为是方块效应，则可能造成新的误差。为此，在滤波方块效应时，应该先判断该边界是图像真实边界还是方块效应所形成的边界（假边界）。对真实边界不进行滤波处理，而对假边界则要根据周围图像块的性质和编码方法采用不同强度的滤波。

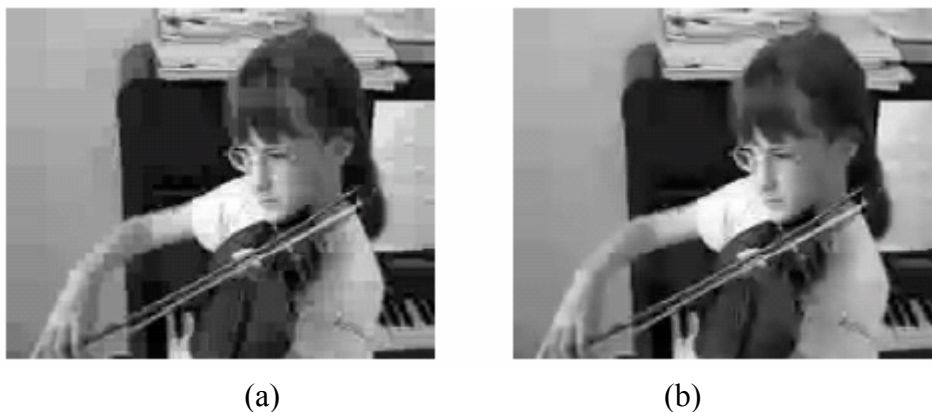


图 6.63 不采用(a)和采用(b)去方块滤波器的 H.264 编解码器的效果

6.11.2 边界分析

6.11.2.1 4×4 方块的误差分析

当对残差用方块变换进行编码时，方块边界比内部的编码误差大。对这个现象的合理解释是内部点的重建是对周围点进行加权平均得到。而边界点所用到的加权平均点较少，所以重建效果较差。这个误差分布不均匀的后果是需要方块边界滤波以提高图像客观质量。

H.264/MPEG-4 AVC 去方块滤波器能适应以下不同水平的需要：

- 片级，全局滤波强度能根据视频序列个体特征进行调节。
- 图像块边界级，滤波强度依赖于边界两边图像块的帧间/帧内预测、运动矢量差别及变换量化是否是对残差编码的。特别强的滤波应用于非常平坦的图像宏块以去除“马赛克效应”。
- 图像样点级，样点值及与量化参数相关的阈值可以决定是否对每个样点进行滤波。

下面将讨论 H.264/MPEG-4 AVC 去方块滤波器如何根据这些自适应性设计的细节。

6.11.2.2 自适应边界级滤波器

边界强度（Bs）决定去方块滤波器选择滤波参数，控制去除方块效应的程度。对所有 4×4 亮度块间的边界，边界强度参数值在 0 到 4 之间，它与边界的性质有关。表 6.22 说明 Bs 与相邻图像块的模式及编码条件的关系。表中的条件是从表的上部至下部进行判断的，直到某一条件满足，给 Bs 相应赋值。

表 6.22 滤波器强度参数与编码模式的关系

图像块模式与条件	Bs
边界两边一个图像块为帧内预测并且边界为宏块边界	4
边界两边一个图像块为帧内预测	3
边界两边一个图像块对残差编码	2
边界两边图像块运动矢量差不小于 1 个亮度图像点距离	1
边界两边图像块运动补偿的参考帧不同	1
其它	0

在实际滤波算法中，Bs 决定对边界的滤波强度，包括对两个主要滤波模式的选择。当其值为 4 时表示要用特定最强的滤波模式，而其值为 0 表示不需要对边界进行滤波。对其值为 1 到 3 的标准滤波模式，Bs 值影响滤波器对样点的最大修正程度。Bs 值的下降趋势说明最强的方块效应主要来自于帧内预测模式及对预测残差编码，而在较小程度上与图像的运动补偿有关。

色度块边界滤波的 Bs 值不另外单独计算，而是从相应亮度块边界的 Bs 值复制而来。

在帧场自适应宏块中，表 6.22 中的条件相对复杂些，因为相邻两图像块中的一个可能来自帧编码宏块或来自场编码宏块。滤波强度变化的原则不变。为了避免将图像过度模糊化，对于来自场编码宏块的水平边界需要特别考虑以避免过强的滤波强度，这是因为这种宏块的垂直滤波的空间扩展范围是其它情况的两倍。

6.11.2.3 自适应样点级滤波器

在去方块滤波中，非常重要的是要区分图像中的真实边界和由 DCT 变换系数量化而造成的假边界。为了保持图像的逼真度，应该尽量滤除假边界以不致被看出的同时保持图像真实边界不被滤波。

为了区分这两种情况，要分析每个需要被滤波的边界两边的样点值。这里，定义两个相邻 4×4 块中一条直线上的样点为 p<sub>3</sub>、p<sub>2</sub>、p<sub>1</sub>、p<sub>0</sub>、q<sub>0</sub>、q<sub>1</sub>、q<sub>2</sub>、q<sub>3</sub>，实际的图像边界在 p<sub>0</sub> 和 q<sub>0</sub> 之间，如图 6.64 所示。

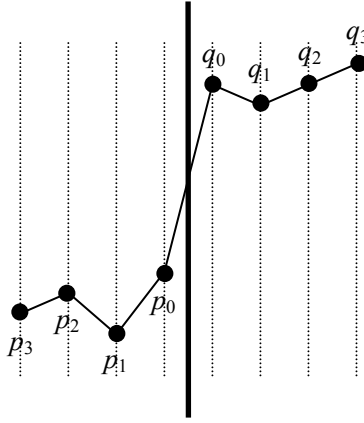


图 6.64 典型的不需要去方块滤波的图像边界

如上小节所述，当  $B_s$  值为 0 时，滤波器对边界不起作用。对于  $B_s$  值为非 0 的边界，为区分上述真假两种边界，定义一对与量化有关的参数，为  $\alpha$  和  $\beta$ ，用来检查图像内容，以决定每个样本点集是否要被滤波。只有下述三个条件同时满足，直线上的样点才被滤波：

$$|p_0 - q_0| < \alpha (Index_A) \quad (6.55)$$

$$|p_1 - p_0| < \beta (Index_B) \quad (6.56)$$

$$|q_1 - q_0| < \beta (Index_B) \quad (6.57)$$

$\alpha$  和  $\beta$  值根据边界两边的平均量化参数查表得到， $\alpha$  和  $\beta$  的查表指数根据下式计算：

$$Index_A = \begin{cases} 0 & QP + Offset_A \leq 0 \\ QP + Offset_A & 0 < QP + Offset_A < 51 \\ 51 & QP + Offset_A \geq 51 \end{cases} \quad (6.58)$$

$$Index_B = \begin{cases} 0 & QP + Offset_B \leq 0 \\ QP + Offset_B & 0 < QP + Offset_B < 51 \\ 51 & QP + Offset_B \geq 51 \end{cases} \quad (6.59)$$

其中，0 到 51 为  $QP$  的范围。 $Offset_A$  和  $Offset_B$  为在编码器中选择的偏移值，以在片级上控制去方块滤波的性能。

$\alpha$  和  $\beta$  值满足下面近似经验关系：

$$\alpha(x) = 0.8(2^{x/6} - 1) \quad (6.60)$$

$$\beta(x) = 0.5x - 7 \quad (6.61)$$

这个关系式中的变量是根据测试进行选择的，让不同的内容得到满意的视觉效果。一般来说  $\beta(x)$  比  $\alpha(x)$  小。为了节省计算量， $\alpha$  和  $\beta$  值通过查找与上式关系一致的表格得到，如表 6.23。特别地，在表格低端，取值被限为 0，这样对  $Index_A < 16$  或  $Index_B < 16$ ， $\alpha$  和  $\beta$  中的一个或两个全部为 0，相应地不进行滤波。

表 6.23 阈值变量 $\alpha$ 和 $\beta$ 与变量 indexA 和 indexB 的关系

		$Index_A$ (对应 $\alpha$ ) 或者 $index_B$ (对应 $\beta$ )																																							
		<16	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36																		
$\alpha$		0	4	4	5	6	7	8	9	10	12	13	15	17	20	22	25	28	32	36	40	45	50																		
$\beta$		0	2	2	2	3	3	3	3	4	4	4	6	6	7	7	8	8	9	9	10	10	11																		
		$Index_A$ (对应 $\alpha$ ) 或者 $index_B$ (对应 $\beta$ )																																							
		37	38	39	40	41	42	43	44	45	46	47	48	49	50	51																									
$\alpha$		56	63	71	80	90	101	113	127	144	162	182	203	226	255	255																									
$\beta$		11	12	12	13	13	14	14	15	15	16	16	17	17	18	18																									

$\alpha$ 和 $\beta$ 与  $QP$  的关系将滤波强度与滤波前重建的图像一般质量联系起来。因为阈值随  $QP$  增加，当  $QP$  较大时，含有较多内容活性的边界需要被滤波，这是由于编码误差随  $QP$  增加。 $\alpha$ 中的指数特性反映期望的方块效应与 $\alpha$ 的关系，因为  $QP$  每增加 6 则量化步长增加一倍。

#### 6.11.2.4 自适应片级滤波器

在片级上，编码器可以选择偏移 ( $Offset_A$  和  $Offset_B$ ) 来调整滤波器中的 $\alpha$ 和 $\beta$ 值，相对于 0 偏移增加或减少滤波强度。偏移值在图像片头通过句法元素传输，应用于根据  $QP$  值查表求 $\alpha$ 和 $\beta$ 值，如式(6.60)和(6.61)。

利用传递非 0 偏移控制去方块滤波性能，使解码器设计者可以优化解码视频质量，获得比默认表（当偏移为 0 时）效果好的图像。例如，通过传递负的偏移以减少滤波强度，可以有助于小的空间细节的逼真度，特别是高分辨率的视频内容，因为这时小的方块效应不容易被觉察。相反，采用正的偏移以增加滤波强度，可以去除默认表所不能滤除的方块效应，增加图像主观质量。这有助于平滑过渡的低分辨率内容的亮度块情况，去除可能由次优的运动估计、模式选择或残差编码引起的额外方块。

### 6.11.3 滤波过程

#### 6.11.3.1 滤波运算概述

为了保证编码器和解码器中的滤波过程完全一致，对每个编码图像的滤波运算必需按规定顺序进行。滤波应该在适当位置上进行，这样边界两边直线上修改过的样点值作为后续运算的输入值而不引入的误差。

滤波是基于宏块基础上的，先对垂直边界进行水平滤波，再对水平边界进行垂直滤波。对宏块的两个方向滤波都完成后才能进行后面宏块的滤波。对图像中宏块的滤波按 raster 扫描方式进行。对帧场自适应编码帧，它们在垂直方向上相邻的宏块对放在一起，则滤波顺序按宏块对进行，即在帧中对宏块对进行按 raster 扫描方式，对每个宏块对先进行顶部宏块的滤波。

对每个亮度宏块，先滤波宏块最左边的边界（如图 6.65 中的 a），然后依次从左到右宏块内三个垂直边界（如图 6.65 中的 b 到 d）。类似的，对水平边界先滤波宏块顶部的边界（如图 6.65 中的 e），然后依次从上到下宏块内三个水平边界（如图 6.65 中的 f 到 h）。色度滤波次序类似，对 8×8 的色度宏块，在每个方向上，先滤波宏块外部边界再滤波一个内部边界（如在图 6.65 中，水平方向先滤波 i，再滤波 j；垂直方向上先滤波 k，再滤波 l）。

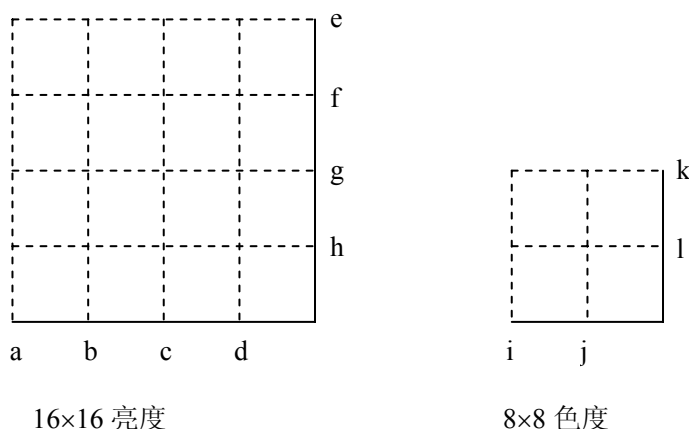


图 6.65 宏块中边界滤波顺序

根据样点集的  $B_s$  值选择两种滤波方式。特定滤波方式是针对  $B_s$  为 4 的强滤波，普通滤波方式应用于其它情况 ( $B_s=1, 2, 3$ )。

对每种方式，用  $\beta$  阈值估计另外两个空间变化条件，以决定亮度点的滤波范围。

$$|p_2 - p_0| < \beta(\text{Index}_B) \quad (6.62)$$

$$|q_2 - q_0| < \beta(\text{Index}_B) \quad (6.63)$$

当上述条件成立，说明边界变化强度不大，滤波强度设定值相对实际滤波来说偏大。

### 6.11.3.2 $B_s$ 值从 1 到 3 的边界滤波

滤波运算可以分为基本滤波运算和限幅两个阶段。

#### (1) 基本滤波运算

先讨论对亮度点的滤波。对这种模式的滤波，滤波后的  $p_0'$  和  $q_0'$  值按下式计算：

$$p_0' = p_0 + \Delta_0 \quad (6.64)$$

$$q_0' = q_0 - \Delta_0 \quad (6.65)$$

其中  $\Delta_0$  分两步计算，先计算它的初始值  $\Delta_{0i}$ ，再对这个初始值进行限幅后代入上式。

初始值  $\Delta_{0i}$  根据边界两边的样点值计算：

$$\Delta_{0i} = (4(q_0 - p_0) + (p_1 - q_1) + 4) \gg 3 \quad (6.66)$$

计算  $p_0'$  的脉冲响应运算为  $(1, 4, 4, -1) / 8$ 。

只有式(6.62)或(6.63)成立，才修正对应  $p_1$  或  $q_1$  值。即如果式(6.62)成立，滤波后的  $p_1'$  值按下式计算：

$$p_1' = p_1 + \Delta_{p1} \quad (6.67)$$

同样，如果式(6.63)成立，滤波后的  $q_1'$  值按下式计算：

$$q_1' = q_1 + \Delta_{q1} \quad (6.68)$$

这些同样要经过两步计算。对  $p_1'$  的初始值  $\Delta$  按下式计算：

$$\Delta_{p1i} = (p_2 + ((p_0 + q_0 + 1) \gg 1) - 2p_1) \gg 1 \quad (6.69)$$

$\Delta_{q1i}$  按同样关系式计算，用  $q_2$  和  $q_1$  分别代替  $p_2$  和  $p_1$ 。上式相应的脉冲响应为  $(1, 0, 0.5, -0.5) / 2$ ，具有很强的低通特性。

#### (2) 限幅

如果上述初始值  $\Delta_{0i}$ 、 $\Delta_{p1i}$  和  $\Delta_{q1i}$  直接应用在滤波计算中，则可能导致滤波频率过低，出现图像模糊。自适应滤波器的一个重要部分是限制  $\Delta$  的值。这个过程称为限幅。对于内部和边界上的样点，限



幅过程不同。

对于内部样点，用于滤波的 $\Delta$ 值被限制在 $-c_1$ 到 $c_1$ 范围内， $c_1$ 是从二维表（表 6.24）中查找的参数，它是根据用于计算 $\alpha$ 的  $Index_A$  和  $Bs$  查找。 $Index_A$  和  $Bs$  越大，则  $c_1$  也越大，允许更强的滤波。最终对  $p_1$  和  $q_1$  滤波的限幅值为：

$$\Delta_{p1} = \begin{cases} -c_1 & \Delta_{pli} \leq -c_1 \\ \Delta_{pli} & -c_1 < \Delta_{pli} < c_1 \\ c_1 & \Delta_{pli} \geq c_1 \end{cases} \quad (6.70)$$

$$\Delta_{q1} = \begin{cases} -c_1 & \Delta_{qli} \leq -c_1 \\ \Delta_{qli} & -c_1 < \Delta_{qli} < c_1 \\ c_1 & \Delta_{qli} \geq c_1 \end{cases} \quad (6.71)$$

对于滤波边界  $p_0$  和  $q_0$  样点， $\Delta_{0i}$  的限幅值由  $c_1$  和式(6.62)或(6.63)决定。先将它的限幅值  $c_0$  定为  $c_1$ 。如果式(6.62) 或(6.63)都成立，说明边界两边内部的变化强度小于 $\beta$ 阈值，需要对边界进行更强的滤波（同时如上述需要对  $p_1$  和/或  $q_1$  样点进行修正）， $c_0$  将增加 1。这样对边界样点的修正值为：

$$\Delta_0 = \begin{cases} -c_0 & \Delta_0 \leq -c_0 \\ \Delta_{0i} & -c_0 < \Delta_{0i} < c_0 \\ c_0 & \Delta_{0i} \geq c_0 \end{cases} \quad (6.72)$$

对色度点滤波，只有  $p_0$  和  $q_0$  才被修正。滤波方法与亮度点一样，只是限幅值  $c_0$  为  $c_1$  加 1。这样对  $Bs$  小于 4 的边界没有必要对色度的式(6.62)或(6.63)进行估计，也不必存取变量  $p_2$  和  $q_2$  值。

表 6.24 滤波限幅变量  $c_1$  值与  $indexA$  和  $Bs$  的关系

	IndexA																					
	<17	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	
Bs=1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	
Bs=2	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	3	3
Bs=3	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	4	
	IndexA																					
	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51							
Bs=1	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13							
Bs=2	3	4	4	5	5	6	7	8	8	10	11	12	13	15	17							
Bs=3	5	6	6	7	8	9	10	11	13	14	16	18	20	23	25							

### 6.11.3.3 $Bs$ 值为 4 的边界滤波

H.264/MPEG-4 AVC 的帧内编码在对同一图像区域编码时倾向采用  $16 \times 16$  亮度预测模式。这会在宏块边界引起小幅度的方块效应。但是由于 Mach band 效应，在这种情况下，即使是很小的强度值差别在视觉上的感觉是陡峭的阶梯。为了消除这种马赛克效应，需要在图像内容平滑的两个宏块边界采用较强的滤波器。

对亮度滤波，根据图像内容判断选择较强的 4 拍或 5 拍滤波器，还是较弱的 3 拍滤波器。4 拍或 5 拍滤波器对边界两边的边界点及两个内部点进行修正，而 3 拍滤波器仅改变边界点。只有下面的跨边界差异的约束条件成立才使用较强的滤波器：

$$|p_0 - q_0| < (\alpha > 2) + 2 \quad (6.73)$$

注意，式(6.73)与式(6.53)很相似，只是它跨边界的最大样点值差异的约束很严格。

对亮度滤波，当式(6.62)和(6.73)都成立，根据下式计算滤波后的样点值：

$$p_0' = (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4) \gg 3 \quad (6.74)$$

$$p_1' = (p_2 + p_1 + p_0 + q_0 + 2) \gg 2 \quad (6.75)$$

$$p_2' = (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) \gg 3 \quad (6.76)$$

否则，对色度点或当式(6.62)和(6.73)中只要有一个不成立的亮度点，只根据下式修正  $p_0$ ：

$$p_0' = (2p_1 + p_0 + q_1 + 2) \gg 2 \quad (6.77)$$

$q$  点值的修正方法相同，只是在选择亮度滤波器时用式(6.63)代替式(6.62)。

## 6.12 其余特征

### 6.12.1 参考图像管理

H.264 中，已编码图像存储在编码器和解码器的参考缓冲区（DPB，解码图像缓冲区），并有相应的参考图像列表 list0，以供帧间宏块的运动补偿预测使用。对 B 片预测而言，list0 包含当前图像的前面和后面两个方向的图像，并以显示次序排列；也可同时包含短期和长期参考图像。这里，已编码图像由编码器重建的标为短期图像或刚刚编码图像，并由其帧号标定。长期参考图像是较早的图像，由 LongTermPicNum 标定，保存在 DPB 中，直到被代替或删除。

当一幅图像在编码器被编码重建或在解码器被解码时，它存放在 DPB 并标定为以下中的一种：

1) “非参考”，不用于进一步的预测；2) 短期参考图像；3) 长期参考图像；4) 直接输出显示。list0 中短期参考图像是按 PicNum（由帧号推出的变量）从高到低的顺序排列，长期参考图像按 LongTermPicNum 从低到高的顺序排列。当新的图像加在短期列表的位置 0 时，剩余的短期图像索引号依次增加。当短期和长期图像号达到参考帧最大数时，最高索引号的图像被移出缓冲区，即实行滑动窗内存控制。该操作使得编码器和解码器保持  $N$  幅短期参考图像，包括一幅当前图像和  $(N-1)$  幅已编码图像。

由编码器发送的自适应内存控制命令用来管理短期和长期参考图像索引。这样，短期图像才可能被指定长期帧索引，短期或长期图像才可能标定为“非参考”。编码器从 list0 中选择参考图像，进行帧间宏块编码。而该参考图像的选择由索引号标志，索引 0 对应于短期部分的第一帧，长期帧索引开始于最后一个短期帧。

参考图像缓冲区通常由编码器发送的 IDR（瞬时解码器刷新）编码图像刷新，IDR 图像一般为 I 片或 SI 片。当接受到 IDR 图像时，解码器立即将缓冲区中的图像标为“非参考”。后继的片进行无图像参考编码。通常，编码视频序列的第一幅图像都是 IDR 图形。

### 6.12.2 重排序

在编码器中，每个已量化变换系数的  $4 \times 4$  块以图 6.66 所示的 zig-zag 顺序映射为一个 16 元素的矩阵。在  $16 \times 16$  帧内模式编码的宏块中，每个  $4 \times 4$  亮度块的 DC 系数首先以图 6.66 所示的顺序扫描。剩余的 15 个 AC 系数从第二个位置开始扫描。类似的，色度的  $2 \times 2$  DC 系数以光栅顺序首先扫描，剩余的 15 个 AC 系数从第二个位置开始扫描。



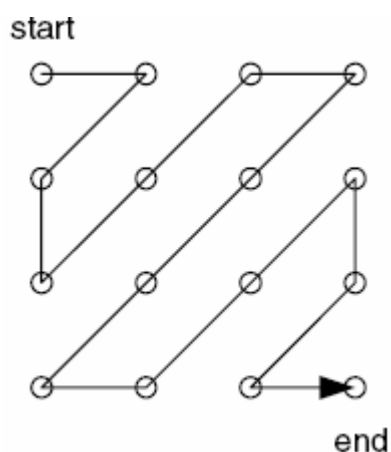


图 6.66 zig-zag 扫描（帧模式）

### 6.12.3 隔行视频

效率高的隔行视频编码工具应该能优化场宏块的压缩。如果场编码被支持，图像的类型（场或帧）在片头中表示。H.264 采用宏块自适应帧场编码（MB-AFF）模式中，帧场编码的选择在宏块级中指定。且当前片通常是 16 亮度像素宽和 32 亮度像素高的单元组成，并以宏块对的形式编码，如图 6.67 所示。编码器可按两个帧宏块或者两个场宏块来编码每个宏块对，也可根据图像每个区域选择最佳编码模式。

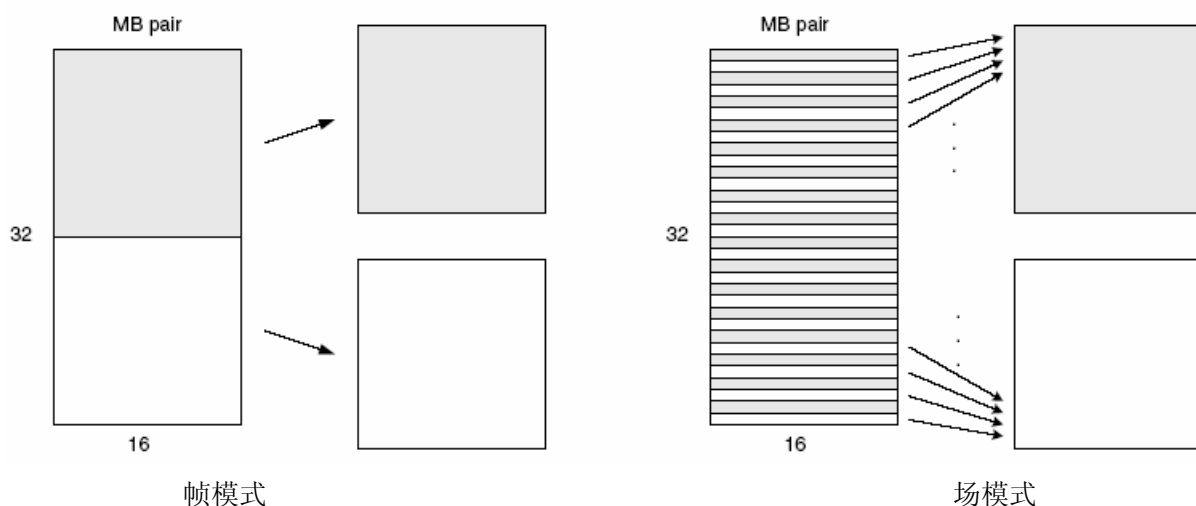


图 6.67 宏块自适应帧场编码

显然，以场模式编码片或宏块对须对编解码的一些步骤进行调整。比如，P 片和 B 片预测中，每个编码场作为一个独立的参考图像；帧内宏块编码模式和帧间宏块 MV 的预测需根据宏块类型（帧还是场）进行调整；图 6.67 所示的重排序扫描也须按图 6.68 所示的顺序进行。

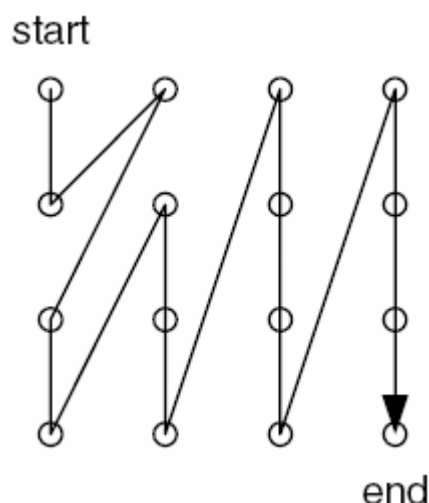


图 6.68 zig-zag 扫描（场模式）

### 6.12.4 数据分割片

组成片的编码数据存放在 3 个独立的 DP（数据分割，A、B、C）中，各自包含一个编码片的子集。分割 A 包含片头和片中每个宏块头数据。分割 B 包含帧内和 SI 片宏块的编码残差数据。分割 C 包含帧间宏块的编码残差数据。每个分割可放在独立的 NAL 单元并独立传输。

如果分割 A 数据丢失，便很难或者不能重建片，因此分割 A 对传输误差很敏感。解码器可根据要求只解 A 和 B 或者 A 和 C，以降低在一定传输条件下的复杂度。

### 6.12.5 H.264 传输

H.264 的编码视频序列包括一系列的 NAL 单元，每个 NAL 单元包含一个 RBSP。如表 6.25 所示。编码片（包括数据分割片和 IDR 片）和序列 RBSP 结束符被定义为 VCL NAL 单元，其余的为 NAL 单元。典型的 RBSP 单元序列如图 6.69 所示。每个单元都按独立的 NAL 单元传送。NAL 单元的头信息（一个字节）定义了 RBSP 单元的类型，NAL 单元的其余部分则为 RBSP 数据。

SPS	SEI	PPS	I 片	图像定界符	P 片	P 片
-----	-----	-----	-----	-------	-----	-----

图 6.69 RBSP 序列举例

表 6.25 RBSP 描述

RBSP 类型	描 述
参数集 PS	序列的全局参数，如图像尺寸、视频格式等等
增强信息 SEI	视频序列解码的增强信息
图像定界符 PD	视频图像的边界
编码片	片的头信息和数据
数据分割	DP 片层的数据，用于错误恢复解码
序列结束符	表明下一图像为 IDR 图像
流结束符	表明该流中已没有图像
填充数据	哑元数据，用于填充字节

#### 1) 参数集

H.264 引入了参数集的概念，每个参数集包含了相应的编码图像的信息。序列参数集 SPS 包含的是针对一连续编码视频序列的参数，如标识符 seq\_parameter\_set\_id、帧数及 POC 的约束、参考帧数目、解码图像尺寸和帧场编码模式选择标识等等。图像参数集 PPS 对应的是一个序列中某一幅图像或者某几幅图像，其参数如标识符 pic\_parameter\_set\_id、可选的 seq\_parameter\_set\_id、熵编码模式选择标识、片组数目、初始量化参数和去方块滤波系数调整标识等等。

通常，SPS 和 PPS 在片的头信息和数据解码前传送至解码器。每个片的头信息对应一个 pic\_parameter\_set\_id，PPS 被其激活后一直有效到下一个 PPS 被激活；类似的，每个 PPS 对应一个 seq\_parameter\_set\_id，SPS 被其激活以后将一直有效到下一个 SPS 被激活。

参数集机制将一些重要的、改变少的序列参数和图像参数与编码片分离，并在编码片之前传送至解码端，或者通过其他机制传输。

## 2) NAL 单元传输和存储

H.264 标准并未定义 NAL 单元的传输方式，但实际中根据不同的传输环境其传输方式还是存在一定的差异。如在包传输网络中，每个 NAL 单元以独立的包传输，在解码之前进行重新排序。在电路交换传输环境中，传输之前须在每个 NAL 单元之前加上起始前缀码，使解码器能够找出 NAL 单元的起始位置。

在一些应用中，编码视频需要和音频及相关信息一起传输存储，这就需要一些机制来实现，目前通常用的是 RTP/UDP 协议协同实现。MPEG-2 System 部分的一个改进版本规定了 H.264 视频传输机制，ITU-T H.241 定义了用 H.264 连接 H.32\*多媒体终端。对要求视频、音频及其他信息一起存储流媒体回放、DVD 回放等应用，将推出的 MPEG-4 System 改进版本定义了 H.264 编码数据和相关媒体流是如何以 ISO 的媒体文件格式存储的。

H.264 提供了许多优化视频编码压缩的机制，并希望能够满足多种媒体通信应用。相对 MPEG-4 来说，H.264 编码工具有一定的限制，但仍可选择多种编码参数和档次。H.264 实际应用是否成功取决于编解码器的设计和编码参数的选择。

## 参考文献

1. T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 560–576, July 2003.
2. S. Wenger, "H.264/AVC over IP," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 645–656, July 2003.
3. T. Stockhammer, M. M. Hannuksela, and T. Wiegand, "H.264/AVC in wireless environments," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 657–673, July 2003.
4. T. Wedi, "Motion compensation in H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 577–586, July 2003.
5. M. Flierl and B. Girod, "Generalized B pictures and the draft JVT/H.264 video compression standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 587–597, July 2003.
6. T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 688–703, July 2003.
7. Schafer Ralf, Wiegand Thomas, Schwarz Heiko. "The emerging H.264/AVC Standard EBU Technical Review", Jan.2003
8. M. Karczewicz and R. Kurçeren, "The SP and SI frames design for H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 637–644, July 2003.
9. H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-Complexity transform and quantization in H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 598–603, July 2003.
10. P. List, A. Joch, J. Lainema, G. Bjøntegaard, and M. Karczewicz, "Adaptive deblocking filter," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 614–619, July 2003.
11. J. Ribas-Corbera, P. A. Chou, and S. Regunathan, "A generalized hypothetical reference decoder for H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 674–687, July 2003.
12. Mathias Wien, "Variable Block-Size Transforms for H.264/AVC" *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 604–613, July 2003.
13. Michael Horowitz, Anthony Joch, Faouzi Kossentini, "H.264/AVC Baseline Profile Decoder Complexity Analysis" *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 704–716, July 2003.
14. T. Wiegand and G. J. Sullivan, "Draft ITU-T Recommendation H.264 and Final Draft International Standard of Joint Video Specification (ITU-T Recommendation H.264 | ISO/IEC 14496-10 AVC)", Joint Video Team of ISO/IEC JTC1/SC29/WG11 and ITU-T SG16/Q.6 Doc. JVT-G050, Pattaya, Thailand, Mar. 2003.
15. JVT Document JVT-C028 G. Bjontegarrd and K. Lillevold, "Context-adaptive VLC Coding of Coefficients", Fairfax, VA, MAY 2002
16. Yao Wang, Jorn Ostermann, Ya-qin Zhang. 视频处理与通信. 北京: 电子工业出版社. 2003。
17. X. L. Ce Zhu, and Lap-Pui Chau. Hexagon-Based Search Pattern for Fast Block Motion Estimation, *IEEE Trans. on CSVT*, vol. Vol.12, 2002: 349-355.
18. Ghanbari, M. The cross-search algorithm for motion estimation, *Communications, IEEE Transactions on*, Volume: 38 Issue:7, July 1990: 950-953.
19. Chang, Y.-C., D. G. Messerschmitt, T. Carney, and S. A. Klein. Delay cognizant video coding: architecture, applications, and quality evaluations. Forthcoming in *IEEE Trans. Image Processing*.
20. Iain E. G. Richardson. H.264 and MPEG-4 Video Compression. Aberdeen, UK. 2003.

21. Joint Model for Non-normative Aspects of Advanced Video Coding. Study of ISO/IEC 14496-6 / PDAM6: 2003 (E).
22. 陈志波 . H.264运动估值与网络视频传输关键问题研究, 清华大学, 毕业论文, 2002.
23. 王汇源 . 数字图像通信原理与技术 . 北京: 国防工业出版社, 2000: 89—92
24. 丁贵广, 计文平, 郭宝龙 , Visual C++6.0 数字图像编码 , 机械工业出版社, 2004。
25. 朱秀昌, 刘峰, 胡栋, 数字图像处理与图像通信, 北京邮电大学出版社,, 2002。
26. Marta KarczewiczThe. SP- and SI- Frames Design for H.264/AVC. IEEE Trans. Circuits and Systems for Video Technology, Vol 13, No.7, July 2003
27. TML 8.7 Software <ftp://standard.pictel.com/video/h261/>
28. 万萍, 陈仁雷, 王海婴, H.264/AVC中的SP/SI帧技术, 电视技术, 2004.1,p22-25

# 第 7 章 H.264 的句法和语义

## 7.1 句法

在编码器输出的码流中，数据的基本单位是句法元素，每个句法元素由若干比特组成，它表示某个特定的物理意义，例如：宏块类型、量化参数等。句法表征句法元素的组织结构，语义阐述句法元素的具体含义。所有的视频编码标准都是通过定义句法和语义来规范编解码器的工作流程。

### 7.1.1 句法元素的分层结构

编码器输出的比特码流中，每个比特都隶属某个句法元素，也就是说，码流是由一个个句法元素依次衔接组成的，码流中除了句法元素并不存在专门用于控制或同步的内容。在 H.264 定义的码流中，句法元素被组织成有层次的结构，分别描述各个层次的信息。图 7.1 表现了这种结构。

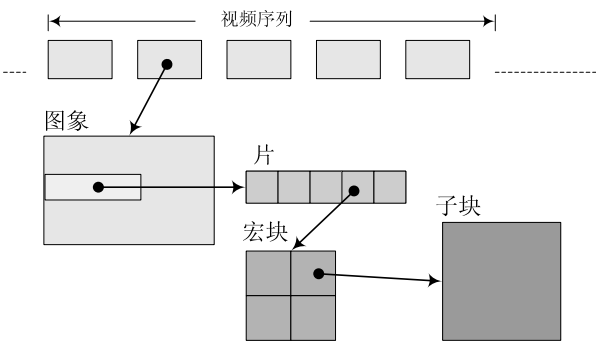


图 7.1 句法元素的分层结构

句法元素的分层结构有助于更有效地节省码流。例如，在一个图像中，经常会在各个片之间有相同的数据，如果每个片都同时携带这些数据，势必会造成码流的浪费。更为有效的做法是将该图像的公共信息抽取出来，形成图像一级的句法元素，而在片级只携带该片自身独有的句法元素。在 H.264 中，句法元素共被组织成 序列、图像、片、宏块、子宏块五个层次。

H.264 的分层结构是经过精心设计的，与以往的视频编码标准相比有很大的改进，这些改进主要针对传输中的错误掩藏，在有误码发生时可以提高图像重建的性能。在以往的标准中，分层的组织结构如图 7.2，它们如同 TCP/IP 协议的结构，每一层都有头部，然后在每层的数据部分包含该层的数据。

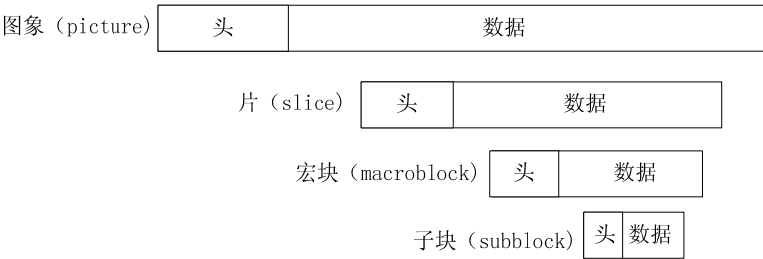


图 7.2 以往标准中句法元素的封装结构

在这样的结构中，每一层的头部和它的数据部分形成管理与被管理的强依赖关系，头部的句法元素是该层数据的核心，而一旦头部丢失，数据部分的信息几乎不可能再被正确解码出来。尤其在序列层及图像层，由于网络中 MTU（最大传输单元）大小的限制，不可能将整个层的句法元素全部放入同一个分组中，这个时候如果头部所在的分组丢失，该层其他分组即使能被正确接收也无法解码，造成资源浪费。

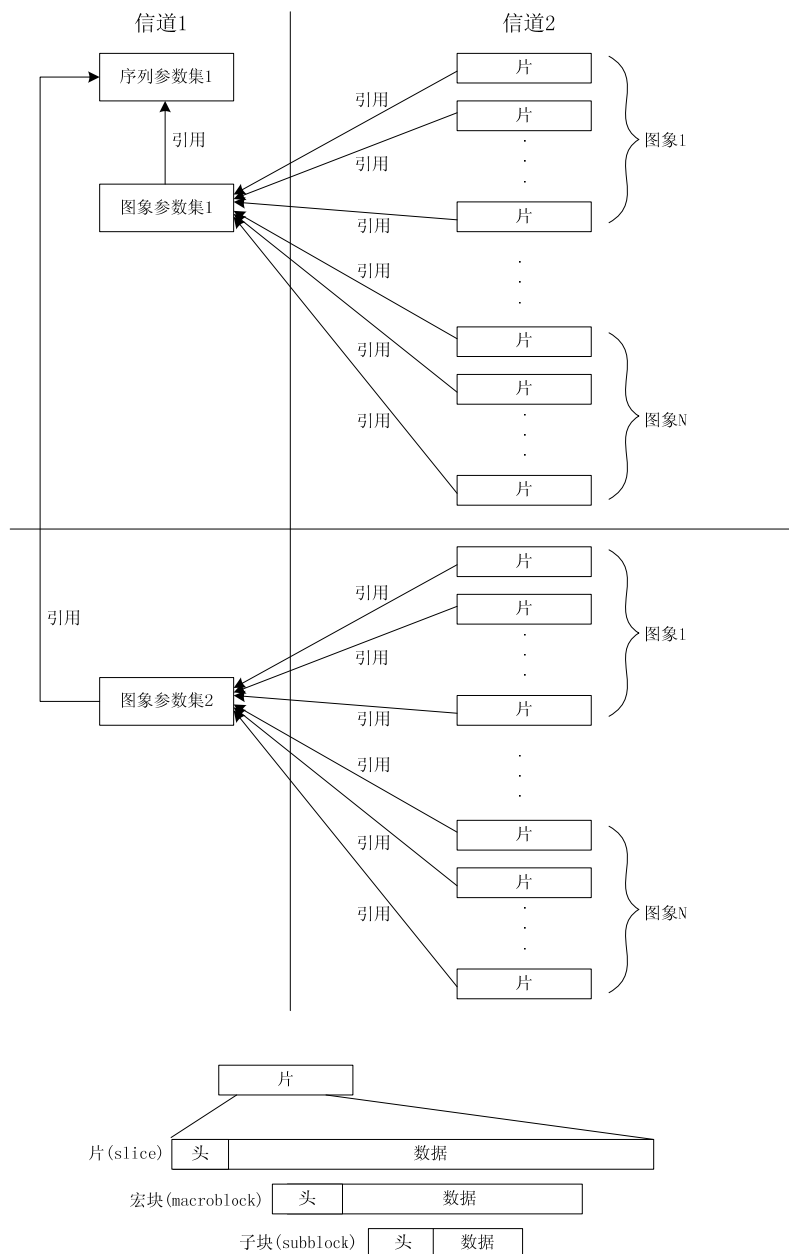


图 7.3 H.264 中句法元素的分层结构

在 H.264 中，分层结构最大的不同是取消了序列层和图像层，并将原本属于序列和图像头部的大部分句法元素游离出来形成序列和图像两级参数集，其余的部分则放入片层。参数集是一个独立的数据单位，不依赖于参数集外的其他句法元素。图 7.3 描述了参数集与参数集外句法元素的关系，在图中我们可以看到，参数集只是在片层句法元素需要的时候被引用，而且，一个参数集并不对应某个特定的图像或序列，同一个序列参数集可以被多个序列中的图像参数集引用，同理，同一个图像参数集也可以被多个图像引用。只在编码器认为需要更新参数集的内容时，才会发送出新的参数集。在这种机制下，由于参数集是独立的，可以被多次重发或者采用特殊技术加以保护。

在图 7.3 的描述中，参数集与参数集外部的句法元素处于不同信道中，这是 H.264 的一个建议，我们可以使用更安全但成本更昂贵的通道来传输参数集，而使用成本低但不够可靠的信道传输其他句法元素，只需要保证片层中的某个句法元素需要引用某个参数集时，那个参数集已经到达解码器，也就是参数集在时间上必须先被传送。当然，在条件不允许的情况下，我们也可以采用妥协的办法：在同一个物理信道中传输所有的句法元素，但专门为参数集采用安全可靠的通信协议，如 TCP。当然，H.264 也允许我们为包括参数集在内的所有句法元素指定同样的通信协议，但这时所有参数集必须被多次重发，以保证解码器最终至少能接收到一个。在参数集和片使用同个物理信道的情况下，图 7.3 中的信道 1 和信道 2 应该被理解为逻辑上的信道，因为从逻辑上看，参数集与其它句法元素还是处于各自彼此独立的信道中。

H.264 在片层增加了新的句法元素指明所引用的参数集的编号，同时因为取消了图像层，片成为了信道 2 中最上层的独立的数据单位，每个片必须自己携带关于所属图像的编号、大小等基本信息，这些信息在同一图像的每个片中都必须是一致的。在编码时，H.264 的规范要求将参数集、片这些独立的数据单位尽可能各自完整地放入一个分组中被传送。

从表面上看来，H.264 关于参数集和片层的结构增加了编码后数据的冗余度（比如参数集必须多次重发，又如每个片都必须携带一部分相同的关于整个图像的信息，而这些数据完全是重复的），降低了编码效率，但这些技术的采用使得通信的鲁棒性大大增强，当数据传输中出现丢包，能够使错误限制在最小范围，防止错误的扩散，解码后对错误的掩藏和恢复也能起到很好的作用。一个片的丢失将不会影响其它片的解码，还可以通过该片前后的片来恢复该片的数据。

H.264 片层以下的句法元素的结构大体上和以往标准类似，但在相当多的细节上有所改进，所有的改进的目的不外乎两个：在错误发生时防止错误扩散、减少冗余信息提高编码效率。这两者往往是矛盾的，H.264 在这两者上的取舍显得颇具匠心，读者在 7.3 语义一节及第八章解码器原理中将会深刻体会到这些。

图 7.3 所示的码流的结构是一种简化的模型，这个模型已经能够正确工作，但还不够完善，不适合复杂的场合。在复杂的通信环境中，除了片和参数集外还需要其他的数据单位来提供额外的信息。图 7.4 描述了在复杂通信中的码流中可能出现的数据单位。如前文所述，参数集可以被抽取出来使用其它信道。

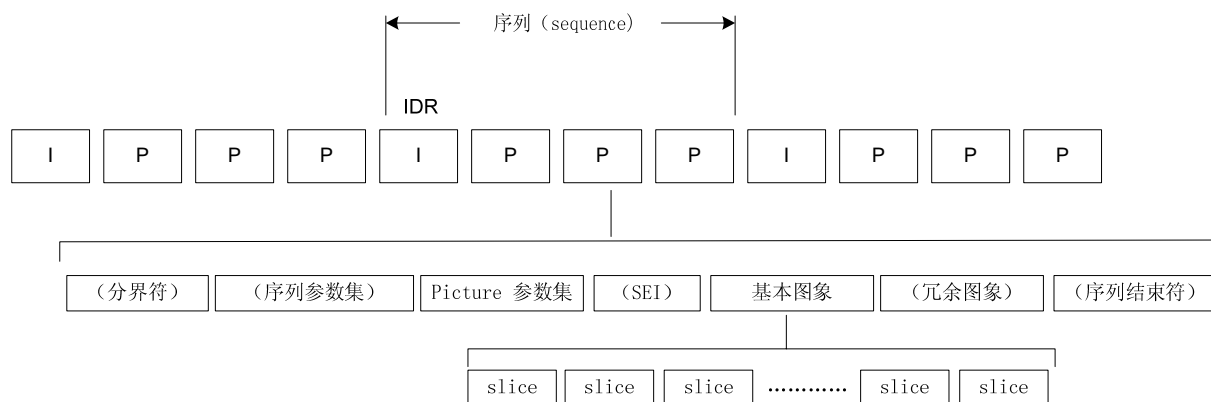


图 7.4 H.264 码流中的数据单位

（这里的数据单位是指可以被独立放入一个分组传输的句法元素集合）

在图 7.4 中我们看到，一个序列的第一个图像叫做 IDR 图像（立即刷新图像），IDR 图像都是 I 图像。H.264 引入 IDR 图像是为了解码的重同步，当解码器解码到 IDR 图像时，立即将参考帧队列清空，将已解码的数据全部输出或抛弃，重新查找参数集，开始一个新的序列。这样，如果在前一个序列的传输中发生重大错误，如严重的丢包，或其他原因引起数据错位，在这里可以获得重新同



步。IDR 图像之后的图像永远不会引用 IDR 图像之前的图像的数据来解码。

要注意 IDR 图像和 I 图像的区别，IDR 图像一定是 I 图像，**但 I 图像不一定是 IDR 图像**。一个序列中可以有很多的 I 图像，I 图像之后的图像可以引用 I 图像之间的图像做运动参考。

在图 7.4 中，除了参数集与片外还有其它的数据单位，这些数据单位可以提供额外的数据或同步信息，这些数据单位也是一系列句法元素的集合。它们在解码过程中不是必需的，但却可以适当提高同步性能或定义图像的复杂特征。

## 7.1.2 句法的表示方法

### 7.1.2.1 句法元素与变量

编码器将数据编码为句法元素然后依次发送。在解码器端，通常要将句法元素作求值计算，得出一些中间数据，这些中间数据就是 H.264 定义的变量。如图 7.5：

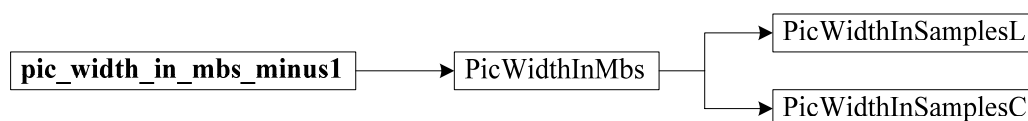


图 7.5 从句法元素解出变量

图中，`pic_width_in_mbs_minus1` 是解码器直接从码流中提取的句法元素，这个句法元素表征图像的宽度，以宏块为单位。我们看到，为了提高编码效率，H.264 将图像实际的宽度减去 1 后再传送。

$$\text{PicWidthInMbs} = \text{pic\_width\_in\_mbs\_minus1} + 1$$

$$\text{PicWidthInSamplesL} = \text{PicWidthInMbs} * 16$$

$$\text{PicWidthInSamplesC} = \text{PicWidthInMbs} * 8$$

以上变量 `PicWidthInMbs` 表示图像以宏块为单位的宽，变量 `PicWidthInSamplesL`、`PicWidthInSamplesC` 分别表示图像的亮度、色度分量以像素为单位的宽。H.264 定义这些变量是因为在后续句法元素的提取算法或图像的重建中需要用到它们的值。**在 H.264 中，句法元素的名称是由小写字母和一系列的下划线组成，而变量名称是大小写字母组成，中间没有下划线。**

### 7.1.2.2 语法

句法是句法元素的组织结构，而对一个结构的描述必然少不了对应的语法，语法提供判断、循环等必要的描述方法。**H.264 采用一种类 C 语法。**

#### 7.1.2.2.1 判断

```
if( 条件 )
{
    ...
}
else
{
    ...
}
```

#### 7.1.2.2.2 循环

与 C 语言类似，H.264 有三种循环体：

a)

```
do
{
    ...
```

```

    }while( 条件 )
b)
    while( 条件 )
    {
        ...
    }
c)
    for( 初始 ; 条件 ; 求值 )
    {
        ...
    }

```

### 7.1.2.3 描述子

**描述子**是指从比特流提取句法元素的方法，即句法元素的解码算法，每个句法元素都有相对应的描述子。由于 H.264 编码的最后一步是熵编码，所以这里的描述子大多是熵编码的解码算法。H.264 定义了如下几种描述子：

- a) **ae(v)** 基于上下文自适应的二进制算术熵编码
- b) **b(8)** 读进连续的 8 个比特
- c) **ce(v)** 基于上下文自适应的可变长熵编码
- d) **f(n)** 读进连续的 n 个比特
- e) **i(n)/i(v)** 读进连续的若干比特，并把它们解释为有符号整数
- f) **me(v)** 映射指数 Golomb 熵编码
- g) **se(v)** 有符号指数 Golomb 熵编码
- h) **te(v)** 截断指数 Golomb 熵编码
- i) **u(n)/u(v)** 读进连续的若干比特，并将它们解释为无符号整数
- j) **ue(v)** 无符号指数 Golomb 熵编码

我们看到，描述子都在括号中带有一个参数，这个参数表示需要提取的比特数。当参数是 **n** 时，表明调用这个描述子的时候会指明 **n** 的值，也即该句法元素是定长编码的。当参数是 **v** 时，对应的句法元素是变长编码，这时有两种情况：**i(v)** 和 **u(v)** 两个描述子的 **v** 由以前的句法元素指定，也就是说在前面会有句法元素指定当前句法元素的比特长度；除了这两个描述子外，其它描述子都是熵编码，它们的解码算法本身能够确定当前句法元素的比特长度。

## 7.2 句法表

句法表定义了 H.264 的句法，指明在码流中依次出现的句法元素及它们出现的条件、提取描述子等。就象前文所提，句法表是分层嵌套的。

在句法表中的 **C** 字段表示该句法元素的分类，这是为片分区服务的，句法元素分类的具体含义在表 7.20 详细介绍。**Descriptor** 指定对应句法元素的描述子。

表 7.1 NAL 层句法

nal_unit( NumBytesInNALunit ) {	C	Descriptor
<b>forbidden_zero_bit</b>	All	f(1)
<b>nal_ref_idc</b>	All	u(2)
<b>nal_unit_type</b>	All	u(5)
NumBytesInRBSP = 0		
for( i = 1; i < NumBytesInNALunit; i++ ) {		
if( i + 2 < NumBytesInNALunit && next_bits( 24 ) == 0x000003 ) {		
<b>rbsp_byte</b> [ NumBytesInRBSP++ ]	All	b(8)
<b>rbsp_byte</b> [ NumBytesInRBSP++ ]	All	b(8)
i += 2		
<b>emulation_prevention_three_byte</b> /* equal to 0x03 */	All	f(8)
} else		
<b>rbsp_byte</b> [ NumBytesInRBSP++ ]	All	b(8)
}		
}		

表 7.2 序列参数集层句法

seq_parameter_set_rbsp( ) {	<b>C</b>	<b>Descriptor</b>
<b>profile_idc</b>	0	u(8)
<b>constraint_set0_flag</b>	0	u(1)
<b>constraint_set1_flag</b>	0	u(1)
<b>constraint_set2_flag</b>	0	u(1)
<b>reserved_zero_5bits</b> /* equal to 0 */	0	u(5)
<b>level_idc</b>	0	u(8)
<b>seq_parameter_set_id</b>	0	ue(v)
<b>log2_max_frame_num_minus4</b>	0	ue(v)
<b>pic_order_cnt_type</b>	0	ue(v)
if( pic_order_cnt_type == 0 )		
<b>log2_max_pic_order_cnt_lsb_minus4</b>	0	ue(v)
else if( pic_order_cnt_type == 1 ) {		
<b>delta_pic_order_always_zero_flag</b>	0	u(1)
<b>offset_for_non_ref_pic</b>	0	se(v)
<b>offset_for_top_to_bottom_field</b>	0	se(v)
<b>num_ref_frames_in_pic_order_cnt_cycle</b>	0	ue(v)
for( i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++ )		
<b>offset_for_ref_frame[ i ]</b>	0	se(v)
}		
<b>num_ref_frames</b>	0	ue(v)
<b>gaps_in_frame_num_value_allowed_flag</b>	0	u(1)
<b>pic_width_in_mbs_minus1</b>	0	ue(v)
<b>pic_height_in_map_units_minus1</b>	0	ue(v)
<b>frame_mbs_only_flag</b>	0	u(1)
if( !frame_mbs_only_flag )		
<b>mb_adaptive_frame_field_flag</b>	0	u(1)
<b>direct_8x8_inference_flag</b>	0	u(1)
<b>frame_cropping_flag</b>	0	u(1)
if( frame_cropping_flag ) {		
<b>frame_crop_left_offset</b>	0	ue(v)
<b>frame_crop_right_offset</b>	0	ue(v)
<b>frame_crop_top_offset</b>	0	ue(v)
<b>frame_crop_bottom_offset</b>	0	ue(v)
}		
<b>vui_parameters_present_flag</b>	0	u(1)

if( vui_parameters_present_flag )		
vui_parameters( )	0	
rbsp_trailing_bits( )	0	
}		

表 7.3 图像参数集层句法

pic_parameter_set_rbsp( ) {	C	Descriptor
<b>pic_parameter_set_id</b>	1	ue(v)
<b>seq_parameter_set_id</b>	1	ue(v)
<b>entropy_coding_mode_flag</b>	1	u(1)
<b>pic_order_present_flag</b>	1	u(1)
<b>num_slice_groups_minus1</b>	1	ue(v)
if( num_slice_groups_minus1 > 0 ) {		
<b>slice_group_map_type</b>	1	ue(v)
if( slice_group_map_type == 0 )		
for( iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++ )		
<b>run_length_minus1[ iGroup ]</b>	1	ue(v)
else if( slice_group_map_type == 2 )		
for( iGroup = 0; iGroup < num_slice_groups_minus1; iGroup++ ) {		
<b>top_left[ iGroup ]</b>	1	ue(v)
<b>bottom_right[ iGroup ]</b>	1	ue(v)
}		
else if( slice_group_map_type == 3    slice_group_map_type == 4    slice_group_map_type == 5 ) {		
<b>slice_group_change_direction_flag</b>	1	u(1)
<b>slice_group_change_rate_minus1</b>	1	ue(v)
} else if( slice_group_map_type == 6 ) {		
<b>pic_size_in_map_units_minus1</b>	1	ue(v)
for( i = 0; i <= pic_size_in_map_units_minus1; i++ )		
<b>slice_group_id[ i ]</b>	1	u(v)
}		
}		
<b>num_ref_idx_l0_active_minus1</b>	1	ue(v)
<b>num_ref_idx_l1_active_minus1</b>	1	ue(v)

<b>weighted_pred_flag</b>	1	u(1)
<b>weighted_bipred_idc</b>	1	u(2)
<b>pic_init_qp_minus26</b> /* relative to 26 */	1	se(v)
<b>pic_init_qs_minus26</b> /* relative to 26 */	1	se(v)
<b>chroma_qp_index_offset</b>	1	se(v)
<b>deblocking_filter_control_present_flag</b>	1	u(1)
<b>constrained_intra_pred_flag</b>	1	u(1)
<b>redundant_pic_cnt_present_flag</b>	1	u(1)
rbsp_trailing_bits( )	1	
}		

表 7.4 片层句法(不分区)

slice_layer_without_partitioning_rbsp( ) {	<b>C</b>	<b>Descriptor</b>
slice_header( )	2	
slice_data( ) /* all categories of slice_data( ) syntax */	2   3   4	
rbsp_slice_trailing_bits( )	2	
}		

表 7.5 片层 A 分区句法

slice_data_partition_a_layer_rbsp( ) {	<b>C</b>	<b>Descriptor</b>
slice_header( )	2	
<b>slice_id</b>	2	ue(v)
slice_data( ) /* only category 2 parts of slice_data( ) syntax */	2	
rbsp_slice_trailing_bits( )	2	
}		

表 7.6 片层 B 分区句法

slice_data_partition_b_layer_rbsp( ) {	<b>C</b>	<b>Descriptor</b>
<b>slice_id</b>	3	ue(v)
if( redundant_pic_cnt_present_flag )		
<b>redundant_pic_cnt</b>	3	ue(v)
slice_data( ) /* only category 3 parts of slice_data( ) syntax */	3	
rbsp_slice_trailing_bits( )	3	
}		

表 7.7 片层 C 分区句法

slice_data_partition_c_layer_rbsp( ) {	<b>C</b>	<b>Descriptor</b>
<b>slice_id</b>	4	ue(v)
if( redundant_pic_cnt_present_flag )		
<b>redundant_pic_cnt</b>	4	ue(v)
slice_data( ) /* only category 4 parts of slice_data( ) syntax */	4	
rbsp_slice_trailing_bits( )	4	
}		

表 7.8 拖尾（trailing bits）句法

rbbsp_trailing_bits( ) {	<b>C</b>	<b>Descriptor</b>
<b>rbbsp_stop_one_bit</b> /* equal to 1 */	All	f(1)
while( !byte_aligned( ) )		
<b>rbbsp_alignment_zero_bit</b> /* equal to 0 */	All	f(1)
}		

表 7.9 片头句法

slice_header( ) {	<b>C</b>	<b>Descriptor</b>
<b>first_mb_in_slice</b>	2	ue(v)
<b>slice_type</b>	2	ue(v)
<b>pic_parameter_set_id</b>	2	ue(v)
<b>frame_num</b>	2	u(v)
if( !frame_mbs_only_flag ) {		
<b>field_pic_flag</b>	2	u(1)
if( field_pic_flag )		
<b>bottom_field_flag</b>	2	u(1)
}		
if( nal_unit_type == 5 )		
<b>idr_pic_id</b>	2	ue(v)
if( pic_order_cnt_type == 0 ) {		
<b>pic_order_cnt_lsb</b>	2	u(v)
if( pic_order_present_flag && !field_pic_flag )		
<b>delta_pic_order_cnt_bottom</b>	2	se(v)
}		
if( pic_order_cnt_type == 1 && !delta_pic_order_always_zero_flag ) {		
<b>delta_pic_order_cnt[ 0 ]</b>	2	se(v)
if( pic_order_present_flag && !field_pic_flag )		
<b>delta_pic_order_cnt[ 1 ]</b>	2	se(v)
}		
if( redundant_pic_cnt_present_flag )		
<b>redundant_pic_cnt</b>	2	ue(v)
if( slice_type == B )		
<b>direct_spatial_mv_pred_flag</b>	2	u(1)
if( slice_type == P    slice_type == SP    slice_type == B ) {		
<b>num_ref_idx_active_override_flag</b>	2	u(1)
if( num_ref_idx_active_override_flag ) {		
<b>num_ref_idx_l0_active_minus1</b>	2	ue(v)
if( slice_type == B )		
<b>num_ref_idx_l1_active_minus1</b>	2	ue(v)
}		
}		
ref_pic_list_reordering( )	2	



if( ( weighted_pred_flag && ( slice_type == P    slice_type == SP ) )    ( weighted_bipred_idc == 1 && slice_type == B ) )		
pred_weight_table( )	2	
if( nal_ref_idc != 0 )		
dec_ref_pic_marking( )	2	
if( entropy_coding_mode_flag && slice_type != I && slice_type != SI )		
<b>cabac_init_idc</b>	2	ue(v)
<b>slice_qp_delta</b>	2	se(v)
if( slice_type == SP    slice_type == SI ) {		
if( slice_type == SP )		
<b>sp_for_switch_flag</b>	2	u(1)
<b>slice_qs_delta</b>	2	se(v)
}		
if( deblocking_filter_control_present_flag ) {		
<b>disable_deblocking_filter_idc</b>	2	ue(v)
if( disable_deblocking_filter_idc != 1 ) {		
<b>slice_alpha_c0_offset_div2</b>	2	se(v)
<b>slice_beta_offset_div2</b>	2	se(v)
}		
}		
if( num_slice_groups_minus1 > 0 && slice_group_map_type >= 3 && slice_group_map_type <= 5 )		
<b>slice_group_change_cycle</b>	2	u(v)
}		

表 7.10 参考帧队列重排序（reordering）句法

ref_pic_list_reordering() {	<b>C</b>	<b>Descriptor</b>
if( slice_type != I && slice_type != SI ) {		
<b>ref_pic_list_reordering_flag_l0</b>	2	u(1)
if( ref_pic_list_reordering_flag_l0 )		
do {		
<b>reordering_of_pic_nums_idc</b>	2	ue(v)
if( reordering_of_pic_nums_idc == 0    reordering_of_pic_nums_idc == 1 )		
<b>abs_diff_pic_num_minus1</b>	2	ue(v)
else if( reordering_of_pic_nums_idc == 2 )		
<b>long_term_pic_num</b>	2	ue(v)
} while( reordering_of_pic_nums_idc != 3 )		
}		
if( slice_type == B ) {		
<b>ref_pic_list_reordering_flag_l1</b>	2	u(1)
if( ref_pic_list_reordering_flag_l1 )		
do {		
<b>reordering_of_pic_nums_idc</b>	2	ue(v)
if( reordering_of_pic_nums_idc == 0    reordering_of_pic_nums_idc == 1 )		
<b>abs_diff_pic_num_minus1</b>	2	ue(v)
else if( reordering_of_pic_nums_idc == 2 )		
<b>long_term_pic_num</b>	2	ue(v)
} while( reordering_of_pic_nums_idc != 3 )		
}		
}		

表 7.11 加权预测句法

pred_weight_table( ) {	<b>C</b>	<b>Descriptor</b>
<b>luma_log2_weight_denom</b>	2	ue(v)
<b>chroma_log2_weight_denom</b>	2	ue(v)
for( i = 0; i <= num_ref_idx_l0_active_minus1; i++ ) {		
<b>luma_weight_l0_flag</b>	2	u(1)
if( luma_weight_l0_flag ) {		
<b>luma_weight_l0[ i ]</b>	2	se(v)
<b>luma_offset_l0[ i ]</b>	2	se(v)
}		
<b>chroma_weight_l0_flag</b>	2	u(1)
if( chroma_weight_l0_flag )		
for( j = 0; j < 2; j++ ) {		
<b>chroma_weight_l0[ i ][ j ]</b>	2	se(v)
<b>chroma_offset_l0[ i ][ j ]</b>	2	se(v)
}		
}		
if( slice_type == B )		
for( i = 0; i <= num_ref_idx_l1_active_minus1; i++ ) {		
<b>luma_weight_l1_flag</b>	2	u(1)
if( luma_weight_l1_flag ) {		
<b>luma_weight_l1[ i ]</b>	2	se(v)
<b>luma_offset_l1[ i ]</b>	2	se(v)
}		
<b>chroma_weight_l1_flag</b>	2	u(1)
if( chroma_weight_l1_flag )		
for( j = 0; j < 2; j++ ) {		
<b>chroma_weight_l1[ i ][ j ]</b>	2	se(v)
<b>chroma_offset_l1[ i ][ j ]</b>	2	se(v)
}		
}		
}		
}		

表 7.12 参考帧队列标记(marking)句法

dec_ref_pic_marking( ) {	C	Descriptor
if( nal_unit_type == 5 ) {		
<b>no_output_of_prior_pics_flag</b>	2   5	u(1)
<b>long_term_reference_flag</b>	2   5	u(1)
} else {		
<b>adaptive_ref_pic_marking_mode_flag</b>	2   5	u(1)
if( adaptive_ref_pic_marking_mode_flag )		
do {		
<b>memory_management_control_operation</b>	2   5	ue(v)
if( memory_management_control_operation == 1    memory_management_control_operation == 3 )		
<b>difference_of_pic_nums_minus1</b>	2   5	ue(v)
if( memory_management_control_operation == 2 )		
<b>long_term_pic_num</b>	2   5	ue(v)
if( memory_management_control_operation == 3    memory_management_control_operation == 6 )		
<b>long_term_frame_idx</b>	2   5	ue(v)
if( memory_management_control_operation == 4 )		
<b>max_long_term_frame_idx_plus1</b>	2   5	ue(v)
} while( memory_management_control_operation != 0 )		
}		
}		

表 7.13 片层数据句法

slice_data() {	<b>C</b>	<b>Descriptor</b>
if( entropy_coding_mode_flag )		
while( !byte_aligned( ) )		
<b>cabac_alignment_one_bit</b>	2	f(1)
CurrMbAddr = first_mb_in_slice * ( 1 + MbaffFrameFlag )		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if( slice_type != I && slice_type != SI )		
if( !entropy_coding_mode_flag ) {		
<b>mb_skip_run</b>	2	ue(v)
prevMbSkipped = ( mb_skip_run > 0 )		
for( i=0; i<mb_skip_run; i++ )		
CurrMbAddr = NextMbAddress( CurrMbAddr )		
moreDataFlag = more_rbsp_data( )		
} else {		
<b>mb_skip_flag</b>	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if( moreDataFlag ) {		
if( MbaffFrameFlag && ( CurrMbAddr % 2 == 0    ( CurrMbAddr % 2 == 1 && prevMbSkipped ) ) )		
<b>mb_field_decoding_flag</b>	2	u(1)   ae(v)
macroblock_layer( )	2   3   4	
}		
if( !entropy_coding_mode_flag )		
moreDataFlag = more_rbsp_data( )		
else {		
if( slice_type != I && slice_type != SI )		
prevMbSkipped = mb_skip_flag		
if( MbaffFrameFlag && CurrMbAddr % 2 == 0 )		
moreDataFlag = 1		
else {		
<b>end_of_slice_flag</b>	2	ae(v)
moreDataFlag = !end_of_slice_flag		
}		

}		
CurrMbAddr = NextMbAddress( CurrMbAddr )		
} while( moreDataFlag )		
}		

表 7.14 宏块层句法

macroblock_layer() {	<b>C</b>	<b>Descriptor</b>
<b>mb_type</b>	2	ue(v)   ae(v)
if( mb_type == I_PCM ) {		
while( !byte_aligned() )		
<b>pcm_alignment_zero_bit</b>	2	f(1)
for( i = 0; i < 256 * ChromaFormatFactor; i++)		
<b>pcm_byte[ i ]</b>	2	u(8)
} else {		
if( MbPartPredMode( mb_type, 0 ) != Intra_4x4 && MbPartPredMode( mb_type, 0 ) != Intra_16x16 && NumMbPart( mb_type ) == 4 )		
sub_mb_pred( mb_type )	2	
else		
mb_pred( mb_type )	2	
if( MbPartPredMode( mb_type, 0 ) != Intra_16x16 )		
<b>coded_block_pattern</b>	2	me(v)   ae(v)
if( CodedBlockPatternLuma > 0    CodedBlockPatternChroma > 0    MbPartPredMode( mb_type, 0 ) == Intra_16x16 ) {		
<b>mb_qp_delta</b>	2	se(v)   ae(v)
residual( )	3   4	
}		
}		
}		

表 7.15 宏块层预测句法

mb_pred( mb_type ) {	C	Descriptor
if( MbPartPredMode( mb_type, 0 ) == Intra_4x4    MbPartPredMode( mb_type, 0 ) == Intra_16x16 ) {		
if( MbPartPredMode( mb_type, 0 ) == Intra_4x4 )		
for( luma4x4BlkIdx=0; luma4x4BlkIdx<16; luma4x4BlkIdx++ ) {		
<b>prev_intra4x4_pred_mode_flag</b> [ luma4x4BlkIdx ]	2	u(1)   ae(v)
if( !prev_intra4x4_pred_mode_flag[ luma4x4BlkIdx ] )		
<b>rem_intra4x4_pred_mode</b> [ luma4x4BlkIdx ]	2	u(3)   ae(v)
}		
<b>intra_chroma_pred_mode</b>	2	ue(v)   ae(v)
} else if( MbPartPredMode( mb_type, 0 ) != Direct ) {		
for( mbPartIdx = 0; mbPartIdx < NumMbPart( mb_type ); mbPartIdx++)		
if( ( num_ref_idx_l0_active_minus1 > 0    mb_field_decoding_flag ) && MbPartPredMode( mb_type, mbPartIdx ) != Pred_L1 )		
<b>ref_idx_l0</b> [ mbPartIdx ]	2	te(v)   ae(v)
for( mbPartIdx = 0; mbPartIdx < NumMbPart( mb_type ); mbPartIdx++)		
if( ( num_ref_idx_l1_active_minus1 > 0    mb_field_decoding_flag ) && MbPartPredMode( mb_type, mbPartIdx ) != Pred_L0 )		
<b>ref_idx_l1</b> [ mbPartIdx ]	2	te(v)   ae(v)
for( mbPartIdx = 0; mbPartIdx < NumMbPart( mb_type ); mbPartIdx++)		
if( MbPartPredMode( mb_type, mbPartIdx ) != Pred_L1 )		
for( compIdx = 0; compIdx < 2; compIdx++ )		
<b>mvd_l0</b> [ mbPartIdx ][ 0 ][ compIdx ]	2	se(v)   ae(v)
for( mbPartIdx = 0; mbPartIdx < NumMbPart( mb_type ); mbPartIdx++)		
if( MbPartPredMode( mb_type, mbPartIdx ) != Pred_L0 )		
for( compIdx = 0; compIdx < 2; compIdx++ )		
<b>mvd_l1</b> [ mbPartIdx ][ 0 ][ compIdx ]	2	se(v)   ae(v)
}		
}		

表 7.16 子宏块预测句法

sub_mb_pred( mb_type ) {	<b>C</b>	<b>Descriptor</b>
for( mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++ )		
<b>sub_mb_type</b> [ mbPartIdx ]	2	ue(v)   ae(v)
for( mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++ )		
if( ( num_ref_idx_l0_active_minus1 > 0    mb_field_decoding_flag ) && mb_type != P_8x8ref0 && sub_mb_type[ mbPartIdx ] != B_Direct_8x8 && SubMbPredMode( sub_mb_type[ mbPartIdx ] ) != Pred_L1 )		
<b>ref_idx_l0</b> [ mbPartIdx ]	2	te(v)   ae(v)
for( mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++ )		
if( ( num_ref_idx_l1_active_minus1 > 0    mb_field_decoding_flag ) && sub_mb_type[ mbPartIdx ] != B_Direct_8x8 && SubMbPredMode( sub_mb_type[ mbPartIdx ] ) != Pred_L0 )		
<b>ref_idx_l1</b> [ mbPartIdx ]	2	te(v)   ae(v)
for( mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++ )		
if( sub_mb_type[ mbPartIdx ] != B_Direct_8x8 && SubMbPredMode( sub_mb_type[ mbPartIdx ] ) != Pred_L1 )		
for( subMbPartIdx = 0; subMbPartIdx < NumSubMbPart( sub_mb_type[ mbPartIdx ] ); subMbPartIdx++ )		
for( compIdx = 0; compIdx < 2; compIdx++ )		
<b>mvd_l0</b> [ mbPartIdx ][ subMbPartIdx ][ compIdx ]	2	se(v)   ae(v)
for( mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++ )		
if( sub_mb_type[ mbPartIdx ] != B_Direct_8x8 && SubMbPredMode( sub_mb_type[ mbPartIdx ] ) != Pred_L0 )		
for( subMbPartIdx = 0; subMbPartIdx < NumSubMbPart( sub_mb_type[ mbPartIdx ] ); subMbPartIdx++ )		
for( compIdx = 0; compIdx < 2; compIdx++ )		
<b>mvd_l1</b> [ mbPartIdx ][ subMbPartIdx ][ compIdx ]	2	se(v)   ae(v)
}		

表 7.17 残差句法



residual( ) {	C	Descriptor
if( !entropy_coding_mode_flag )		
residual_block = residual_block_cavlc		
else		
residual_block = residual_block_cabac		
if( MbPartPredMode( mb_type, 0 ) == Intra_16x16 )		
residual_block( Intra16x16DCLevel, 16 )	3	
for( i8x8 = 0; i8x8 < 4; i8x8++ ) /* each luma 8x8 block */		
for( i4x4 = 0; i4x4 < 4; i4x4++ ) /* each 4x4 sub-block of block */		
if( CodedBlockPatternLuma & ( 1 << i8x8 ) ) {		
if( MbPartPredMode( mb_type, 0 ) == Intra_16x16 )		
residual_block( Intra16x16ACLevel[ i8x8 * 4 + i4x4 ], 15 )	3	
else		
residual_block( LumaLevel[ i8x8 * 4 + i4x4 ], 16 )	3   4	
} else {		
if( MbPartPredMode( mb_type, 0 ) == Intra_16x16 )		
for( i = 0; i < 15; i++ )		
Intra16x16ACLevel[ i8x8 * 4 + i4x4 ][ i ] = 0		
else		
for( i = 0; i < 16; i++ )		
LumaLevel[ i8x8 * 4 + i4x4 ][ i ] = 0		
}		
for( iCbCr = 0; iCbCr < 2; iCbCr++ )		
if( CodedBlockPatternChroma & 3 ) /* chroma DC residual present */		
residual_block( ChromaDCLevel[ iCbCr ], 4 )	3   4	
else		
for( i = 0; i < 4; i++ )		
ChromaDCLevel[ iCbCr ][ i ] = 0		
for( iCbCr = 0; iCbCr < 2; iCbCr++ )		
for( i4x4 = 0; i4x4 < 4; i4x4++ )		
if( CodedBlockPatternChroma & 2 )		
/* chroma AC residual present */		
residual_block( ChromaACLevel[ iCbCr ][ i4x4 ], 15 )	3   4	
else		
for( i = 0; i < 15; i++ )		
ChromaACLevel[ iCbCr ][ i4x4 ][ i ] = 0		
}		

表 7.18 CAVLC 残差句法

residual_block_cavlc( coeffLevel, maxNumCoeff ) {	<b>C</b>	<b>Descriptor</b>
for( i = 0; i < maxNumCoeff; i++ )		
coeffLevel[ i ] = 0		
<b>coeff_token</b>	3   4	ce(v)
if( TotalCoeff( coeff_token ) > 0 ) {		
if( TotalCoeff( coeff_token ) > 10   &&   TrailingOnes( coeff_token ) < 3 )		
suffixLength = 1		
else		
suffixLength = 0		
for( i = 0; i < TotalCoeff( coeff_token ); i++ )		
if( i < TrailingOnes( coeff_token ) ) {		
<b>trailing_ones_sign_flag</b>	3   4	u(1)
level[ i ] = 1 – 2 * trailing_ones_sign_flag		
} else {		
<b>level_prefix</b>	3   4	ce(v)
levelCode = ( level_prefix << suffixLength )		
if( suffixLength > 0        level_prefix >= 14 ) {		
<b>level_suffix</b>	3   4	u(v)
levelCode += level_suffix		
}		
if( level_prefix == 15   &&   suffixLength == 0 )		
levelCode += 15		
if( i == TrailingOnes( coeff_token )   &&   TrailingOnes( coeff_token ) < 3 )		
levelCode += 2		
if( levelCode % 2 == 0 )		
level[ i ] = ( levelCode + 2 ) >> 1		
else		
level[ i ] = ( –levelCode – 1 ) >> 1		
if( suffixLength == 0 )		
suffixLength = 1		
if( Abs( level[ i ] ) > ( 3 << ( suffixLength – 1 ) )   &&   suffixLength < 6 )		
suffixLength++		
}		
if( TotalCoeff( coeff_token ) < maxNumCoeff ) {		
<b>total_zeros</b>	3   4	ce(v)

zerosLeft = total_zeros		
} else		
zerosLeft = 0		
for( i = 0; i < TotalCoeff( coeff_token ) - 1; i++ ) {		
if( zerosLeft > 0 ) {		
<b>run_before</b>	3   4	ce(v)
run[ i ] = run_before		
} else		
run[ i ] = 0		
zerosLeft = zerosLeft - run[ i ]		
}		
run[ TotalCoeff( coeff_token ) - 1 ] = zerosLeft		
coeffNum = -1		
for( i = TotalCoeff( coeff_token ) - 1; i >= 0; i-- ) {		
coeffNum += run[ i ] + 1		
coeffLevel[ coeffNum ] = level[ i ]		
}		
}		
}		

表 7.19 CABAC 残差句法

residual_block_cabac( coeffLevel, maxNumCoeff ) {	<b>C</b>	<b>Descriptor</b>
<b>coded_block_flag</b>	3   4	ae(v)
if( coded_block_flag ) {		
numCoeff = maxNumCoeff		
i = 0		
do {		
<b>significant_coeff_flag[ i ]</b>	3   4	ae(v)
if( significant_coeff_flag[ i ] ) {		
<b>last_significant_coeff_flag[ i ]</b>	3   4	ae(v)
if( last_significant_coeff_flag[ i ] ) {		
numCoeff = i + 1		
for( j = numCoeff; j < maxNumCoeff; j++ )		
coeffLevel[ j ] = 0		
}		
}		
i++		
} while( i < numCoeff-1 )		
<b>coeff_abs_level_minus1[ numCoeff-1 ]</b>	3   4	ae(v)
<b>coeff_sign_flag[ numCoeff-1 ]</b>	3   4	ae(v)
coeffLevel[ numCoeff-1 ] =		
( coeff_abs_level_minus1[ numCoeff-1 ] + 1 ) *		
( 1 - 2 * coeff_sign_flag[ numCoeff-1 ] )		
for( i = numCoeff-2; i >= 0; i-- ) {		
if( significant_coeff_flag[ i ] ) {		
<b>coeff_abs_level_minus1[ i ]</b>	3   4	ae(v)
<b>coeff_sign_flag[ i ]</b>	3   4	ae(v)
coeffLevel[ i ] = ( coeff_abs_level_minus1[ i ] + 1 ) *		
( 1 - 2 * coeff_sign_flag[ i ] )		
} else		
coeffLevel[ i ] = 0		
}		
} else		
for( i = 0; i < maxNumCoeff; i++ )		
coeffLevel[ i ] = 0		
}		

### 7.3 语义

本节将对句法表中的句法元素作详细解释。

7.3.1 NAL 层语义

在网络传输的环境下，编码器将每个 NAL 各自独立、完整地放入一个分组，由于分组都有头部，解码器可以很方便地检测出 NAL 的分界，依次取出 NAL 进行解码。为了节省码流，H.264 没有另外在 NAL 的头部设立表示起始的句法元素，我们从表 7.1 可以看到这点。但是如果编码数据是储存在介质（如 DVD 光盘）上，由于 NAL 是依次紧密排列，解码器将无法在数据流中分辨每个 NAL 的起始和终止，所以必须要有另外的机制来解决这个问题。

针对这个问题，H.264 草案的附录 B 中指明了一种简单又高效的方案。当数据流是存储在介质上时，在每个 NAL 前添加起始码：

0x000001

在某些类型的介质上，为了寻址的方便，要求数据流在长度上对齐，或必须是某个常数的倍数。考虑到这种情况，H.264 建议在起始码前添加若干字节的 0 来填充，直到该 NAL 的长度符合要求。

在这样的机制下，解码器在码流中检测起始码，作为一个 NAL 的起始标识，当检测到下一个起始码时当前 NAL 结束。H.264 规定当检测到 0x000000 时也可以表征当前 NAL 的结束，这是因为连着的三个字节的 0 中的任何一个字节的 0 要么属于起始码要么是起始码前面添加的 0。

添加起始码是一个解决问题的很好的方法，但上面关于起始码的介绍还不完整，因为忽略了一个重要的问题：如果在 NAL 内部出现了 0x000001 或是 0x000000 的序列怎么办？毫无疑问这种情况是致命的，解码器将把这些本来不是起始码的字节序列当作起始码，而错误地认为这里往后是一个新的 NAL 的开始，进而造成解码数据的错位！而我们做的大量实验证明，NAL 内部经常会出现这样的字节序列。

于是 H.264 提出了另外一种机制，叫做“防止竞争”，在编码器编码完一个 NAL 时，应该检测是否出现图 7.6 左侧 中的四个字节序列，以防止它们和起始码竞争。如果检测到这些序列存在，编码器将在最后一个字节前插入一个新的字节：0x03，从而使它们变成图 7.6 右测的样子。当解码器在 NAL 内部检测到有 0x000003 的序列时，将把 0x03 抛弃，恢复原始数据。

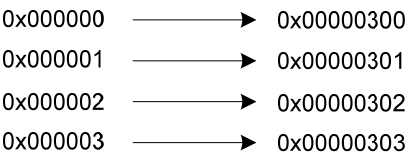


图 7.6 NAL 内部为防止与起始码竞争插入 0x03

图 7.6 中的前两个序列我们前文中已经提到，第三个 0x000002 是作保留用，而第四个 0x000003 是为了保证解码器能正常工作，因为我们刚才提到，解码器恢复原始数据的方法是检测到 0x000003 就抛弃其中的 0x03，这样当出现原始数据为 0x000003 时会破坏数据，所以必须也应该给这个序列插入 0x03。

我们可以从句法表 7.1 中看到解码器在 NAL 层的处理步骤，其中变量 NumBytesInNALunit 是解码器计算出来的，解码器在逐个字节地读一个 NAL 时并不同时对它解码，而是要通过起始码机制将整个 NAL 读进、计算出长度后再开始解码。

**forbidden\_zero\_bit**        等于 0

**nal\_ref\_idc**    指示当前 NAL 的优先级。取值范围为 0-3, 值越高,表示当前 NAL 越重要,需要优先受到保护。H.264 规定如果当前 NAL 是属于参考帧的片，或是序列参数集，或是图像参数集这些重要

的数据单位时，本句法元素必须大于 0。但在大于 0 时具体该取何值，却没有进一步规定,通信双方可以灵活地制定策略。

**nal\_unit\_type** 指明当前 NAL unit 的类型，具体类型的定义如表 7.20:

表 7.20 nal\_unit\_type 语义

nal_unit_type	NAL类型	C
0	未使用	
1	不分区、非 IDR 图像的片	2, 3, 4
2	片分区 A	2
3	片分区 B	3
4	片分区 C	4
5	IDR 图像中的片	2, 3
6	补充增强信息单元 (SEI)	5
7	序列参数集	0
8	图像参数集	1
9	分界符	6
10	序列结束	7
11	码流结束	8
12	填充	9
13..23	保留	
24..31	未使用	

nal\_unit\_type=5 时，表示当前 NAL 是 IDR 图像的一个片，在这种情况下，IDR 图像中的每个片的 nal\_unit\_type 都应该等于 5。注意 IDR 图像不能使用片分区。

**rbsp\_byte[i]** RBSP 的第 i 个字节。RBSP 指原始字节载荷，它是 NAL 单元的数据部分的封装格式，封装的数据来自 SODB（原始数据比特流）。SODB 是编码后的原始数据，SODB 经封装为 RBSP 后放入 NAL 的数据部分。下面介绍一个 RBSP 的生成顺序。

从 SODB 到 RBSP 的生成过程：

- 如果 SODB 内容是空的，生成的 RBSP 也是空的
- 否则，RBSP 由如下的方式生成：
  - 1) RBSP 的第一个字节直接取自 SODB 的第 1 到 8 个比特，（RBSP 字节内的比特按照从左到右对应为从高到低的顺序排列，most significant），以此类推，RBSP 其余的每个字节都直接取自 SODB 的相应比特。RBSP 的最后一个字节包含 SODB 的最后几个比特，及如下的 rbsp\_trailing\_bits()
  - 2) rbsp\_trailing\_bits() 的第一个比特是 1, 接下来填充 0，直到字节对齐。（填充 0 的目的也是为了字节对齐）
  - 3) 最后添加若干个 cabac\_zero\_word(其值等于 0x0000)

**emulation\_prevention\_three\_byte** NAL 内部为防止与起始码竞争而引入的填充字节，值为 0x03。

### 7.3.2 序列参数集语义

**profile\_idc**、**level\_idc** 指明所用 profile、level。

**constraint\_set0\_flag** 等于 1 时表示必须遵从附录 A.2.1 所指明的所有制约条件。等于 0 时表示不必遵从所有条件。

**constraint\_set1\_flag** 等于 1 时表示必须遵从附录 A.2.2 所指明的所有制约条件。等于 0 时表示不必遵从所有条件。

**constraint\_set2\_flag** 等于 1 时表示必须遵从附录 A.2.3 所指明的所有制约条件。等于 0 时表示不必遵从所有条件。

注意：当 **constraint\_set0\_flag**、**constraint\_set1\_flag**、**constraint\_set2\_flag** 中的两个以上等于 1 时，A.2 中的所有制约条件都要被遵从。

**reserved\_zero\_5bits** 在目前的标准中本句法元素必须等于 0，其他的值保留做将来用，解码器应该忽略本句法元素的值。

**seq\_parameter\_set\_id** 指明本序列参数集的 id 号，这个 id 号将被 picture 参数集引用，本句法元素的值应该在[0, 31]。

注意：当编码器需要产生新的序列参数集时，应该使用新的 **seq\_parameter\_set\_id**，即使用新的序列参数集，而不是去改变原来的参数集中的内容

**log2\_max\_frame\_num\_minus4** 这个句法元素主要是为读取另一个句法元素 **frame\_num** 服务的，**frame\_num** 是最重要的句法元素之一，它标识所属图像的解码顺序。可以在句法表看到，**frame-num** 的解码函数是  $ue(v)$ ，函数中的  $v$  在这里指定：

$$v = \text{log2\_max\_frame\_num\_minus4} + 4$$

从另一个角度看，这个句法元素同时也指明了 **frame\_num** 的所能达到的最大值：

$$\text{MaxFrameNum} = 2(\text{log2\_max\_frame\_num\_minus4} + 4)$$

变量 **MaxFrameNum** 表示 **frame\_num** 的最大值，在后文中可以看到，在解码过程中它也是一个非常重要的变量。

值得注意的是 **frame\_num** 是循环计数的，即当它到达 **MaxFrameNum** 后又从 0 重新开始新一轮的计数。解码器必须要有机制检测这种循环，不然会引起类似千年虫的问题，在图像的顺序上造成混乱。在第八章会详细讲述 H.264 检测这种循环的机制。

**pic\_order\_cnt\_type** 指明了 poc (picture order count) 的编码方法，poc 标识图像的播放顺序。由于 H.264 使用了 B 帧预测，使得图像的解码顺序并不一定等于播放顺序，但它们之间存在一定的映射关系。poc 可以由 **frame-num** 通过映射关系计算得来，也可以索性由编码器显式地传送。H.264 中共定义了三种 poc 的编码方法，这个句法元素就是用来通知解码器该用哪种方法来计算 poc。而以下的几个句法元素是分别在各种方法中用到的数据。

在如下的视频序列中本句法元素不应该等于 2：

- 一个非参考帧的接入单元后面紧跟着一个非参考图像(指参考帧或参考场)的接入单元



- 两个分别包含互补非参考场对的接入单元后面紧跟着一个非参考图像的接入单元。
- 一个非参考场的接入单元后面紧跟着另外一个非参考场,并且这两个场不能构成一个互补场对

**log2\_max\_pic\_order\_cnt\_lsb\_minus4** 指明了变量 `MaxPicOrderCntLsb` 的值:

$$\text{MaxPicOrderCntLsb} = 2(\text{log2\_max\_pic\_order\_cnt\_lsb\_minus4} + 4)$$

该变量在 `pic_order_cnt_type = 0` 时使用。

**delta\_pic\_order\_always\_zero\_flag** 等于 1 时,句法元素 `delta_pic_order_cnt[0]`和 `delta_pic_order_cnt[1]`不在片头出现,并且它们的值默认为 0; 本句法元素等于 0 时,上述的两个句法元素将在片头出现。

**offset\_for\_non\_ref\_pic** 被用来计算非参考帧或场的 picture order count (在 8.2.1),本句法元素的值应该在  $[-2^{31}, 2^{31} - 1]$ 。

**offset\_for\_top\_to\_bottom\_field** 被用来计算帧的底场的 picture order count (在 8.2.1), 本句法元素的值应该在  $[-2^{31}, 2^{31} - 1]$ 。

**num\_ref\_frames\_in\_pic\_order\_cnt\_cycle** 被用来解码 picture order count (在 8.2.1),本句法元素的值应该在  $[0, 255]$ 。

**offset\_for\_ref\_frame[i]** 在 picture order count type=1 时用, 用于解码 POC, 本句法元素对循环 `num_ref_frames_in_pic_order_cycle` 中的每一个元素指定一个偏移。

**num\_ref\_frames** 指定参考帧队列可能达到的最大长度, 解码器依照这个句法元素的值开辟存储区, 这个存储区用于存放已解码的参考帧, H.264 规定最多可用 16 个参考帧, 本句法元素的值最大为 16。值得注意的是这个长度以帧为单位, 如果在场模式下, 应该相应地扩展一倍。

**gaps\_in\_frame\_num\_value\_allowed\_flag** 这个句法元素等于 1 时, 表示允许句法元素 `frame_num` 可以不连续。当传输信道堵塞严重时, 编码器来不及将编码后的图像全部发出, 这时允许丢弃若干帧图像。在正常情况下每一帧图像都有依次连续的 `frame_num` 值, 解码器检查到如果 `frame_num` 不连续, 便能确定有图像被编码器丢弃。这时, 解码器必须启动错误掩藏的机制来近似地恢复这些图像, 因为这些图像有可能被后续图像用作参考帧。

当这个句法元素等于 0 时, 表示不允许 `frame_num` 不连续, 即编码器在任何情况下都不能丢弃图像。这时, H.264 允许解码器可以不去检查 `frame_num` 的连续性以减少计算量。这种情况下如果依然发生 `frame_num` 不连续, 表示在传输中发生丢包, 解码器会通过其他机制检测到丢包的发生, 然后启动错误掩藏的恢复图像。

**pic\_width\_in\_mbs\_minus1** 本句法元素加 1 后指明图像宽度, 以宏块为单位:

$$\text{PicWidthInMbs} = \text{pic\_width\_in\_mbs\_minus1} + 1$$

通过这个句法元素解码器可以计算得到亮度分量以像素为单位的图像宽度:

$$\text{PicWidthInSamplesL} = \text{PicWidthInMbs} * 16$$

从而也可以得到色度分量以像素为单位的图像宽度:

$$\text{PicWidthInSamplesC} = \text{PicWidthInMbs} * 8$$

以上变量 `PicWidthInSamplesL`、`PicWidthInSamplesC` 分别表示图像的亮度、色度分量以像素为单位的宽。

H.264 将图像的大小在序列参数集中定义，意味着可以在通信过程中随着序列参数集动态地改变图像的大小，在后文中可以看到，甚至可以将传送的图像剪裁后输出。

**pic\_height\_in\_map\_units\_minus1** 本句法元素加 1 后指明图像高度：

$\text{PicHeightInMapUnits} = \text{pic\_height\_in\_map\_units\_minus1} + 1$

$\text{PicSizeInMapUnits} = \text{PicWidthInMbs} * \text{PicHeightInMapUnits}$

图像的高度的计算要比宽度的计算复杂，因为一个图像可以是帧也可以是场，从这个句法元素可以在帧模式和场模式下分别计算出亮度、色度的高。值得注意的是，这里以 `map_unit` 为单位，`map_unit` 的含义由后文叙述。

**frame\_mbs\_only\_flag** 本句法元素等于 0 时表示本序列中所有图像的编码模式都是帧，没有其他编码模式存在；本句法元素等于 1 时，表示本序列中图像的编码模式可能是帧，也可能是场或帧场自适应，某个图像具体是哪一种要由其他句法元素决定。

结合 `map_unit` 的含义，这里给出上一个句法元素 `pic_height_in_map_units_minus1` 的进一步解析步骤：当 `frame_mbs_only_flag` 等于 1，`pic_height_in_map_units_minus1` 指的是一个 picture 中帧的高度；当 `frame_mbs_only_flag` 等于 0，`pic_height_in_map_units_minus1` 指的是一个 picture 中场的高度，所以可以得到如下以宏块为单位的图像高度：

$\text{FrameHeightInMbs} = (2 - \text{frame\_mbs\_only\_flag}) * \text{PicHeightInMapUnits}$

$\text{PictureHeightInMbs} = (2 - \text{frame\_mbs\_only\_flag}) * \text{PicHeightInMapUnits}$

**mb\_adaptive\_frame\_field\_flag** 指明本序列是否属于帧场自适应模式。`mb_adaptive_frame_field_flag` 等于 1 时表明在本序列中的图像如果不是场模式就是帧场自适应模式，等于 0 时表示本序列中的图像如果不是场模式就是帧模式。。表 列举了一个序列中可能出现的编码模式：

- a. 全部是帧，对应于 `frame_mbs_only_flag=1` 的情况。
- b. 帧和场共存。`frame_mbs_only_flag=0, mb_adaptive_frame_field_flag=0`
- c. 帧场自适应和场共存。`frame_mbs_only_flag=0, mb_adaptive_frame_field_flag=1`

值得注意的是，帧和帧场自适应不能共存在一个序列中。

**direct\_8x8\_inference\_flag** 用于指明 B 片的直接和 skip 模式下运动矢量的预测方法。

**frame\_cropping\_flag** 用于指明解码器是否要将图像裁剪后输出，如果是的话，后面紧跟着的四个句法元素分别指出左右、上下裁剪的宽度。

**frame\_crop\_left\_offset, frame\_crop\_right\_offset, frame\_crop\_bottom\_offset, frame\_crop\_bottom\_offset** 如上一句法元素所述。

**vui\_parameters\_present\_flag** 指明 vui 子结构是否出现在码流中，vui 的码流结构在附录中指明，用以表征视频格式等额外信息。

### 7.3.3 图像参数集语义

**pic\_parameter\_set\_id** 用以指定本参数集的序号，该序号在各片的片头被引用。

**seq\_parameter\_set\_id** 指明本图像参数集所引用的序列参数集的序号。

**entropy\_coding\_mode\_flag** 指明熵编码的选择，本句法元素为 0 时，表示熵编码使用 CAVLC，本句法元素为 1 时表示熵编码使用 CABAC

**pic\_order\_present\_flag** POC 的三种计算方法在片层还各需要用一些句法元素作为参数，本句法元素等于 1 时表示在片头会有句法元素指明这些参数；本句法元素等于 0 时，表示片头不会给出这些参数，这些参数使用默认值。

**num\_slice\_groups\_minus1** 本句法元素加 1 后指明图像中片组的个数。H.264 中没有专门的句法元素用于指明是否使用片组模式，当本句法元素等于 0（即只有一个片组），表示不使用片组模式，后面也不会跟有用于计算片组映射的句法元素。

**slice\_group\_map\_type** 当 num\_slice\_group\_minus1 大于 0，既使用片组模式时，本句法元素出现在码流中，用以指明片组分割类型。

map\_units 的定义：

- 当 frame\_mbs\_only\_flag 等于 1 时，map\_units 指的就是宏块
- 当 frame\_mbs\_only\_flag 等于 0 时
  - 帧场自适应模式时，map\_units 指的是宏块对
  - 场模式时，map\_units 指的是宏块
  - 帧模式时，map\_units 指的是与宏块对相类似的，上下两个连续宏块的组合体。

**run\_length\_minus1[i]** 用以指明当片组类型等于 0 时，每个片组连续的 map\_units 个数。

**top\_left[i],bottom\_right[i]** 用以指明当片组类型等于 2 时，矩形区域的左上及右下位置。

**slice\_group\_change\_direction\_flag** 当片组类型等于 3、4、5 时，本句法元素与下一个句法元素一起指明确切的片组分割方法。

**slice\_group\_change\_rate\_minus1** 用以指明变量 SliceGroupChangeRate

**pic\_size\_in\_map\_units\_minus1** 在片组类型等于 6 时，用以指明图像以 map\_units 为单位的大小。

**slice\_group\_id[i]** 在片组类型等于 6 时，用以指明某个 map\_units 属于哪个片组。

**num\_ref\_idx\_l0\_active\_minus1** 加 1 后指明目前参考帧队列的长度，即有多少个参考帧（包括短期和长期）。值得注意的是，当目前解码图像是场模式下，参考帧队列的长度应该是本句法元素再乘以 2，因为场模式下各帧必须被分解以场对形式存在。（这里所说的场模式包括图像的场及帧场自适应下的处于场模式的宏块对） 本句法元素的值有可能在片头被重载。

读者可能还记得在序列参数集中有句法元素 num\_ref\_frames 也是跟参考帧队列有关，它们的区别是 num\_ref\_frames 指明参考帧队列的最大值，解码器用它的值来分配内存空间；num\_ref\_idx\_l0\_active\_minus1 指明在这个队列中当前实际的、已存在的参考帧数目，这从它的名字“active”中也可以看出来。

这个句法元素是 H.264 中最重要的句法元素之一，在第章我们可以看到，编码器要通知解码器

某个运动矢量所指向的是哪个参考图像时，并不是直接传送该图像的编号，而是传送该图像在参考帧队列中的序号。这个序号并不是在码流中传送的，而是编码器和解码器同步地、用相同的方法将参考图像放入队列，从而获得一个序号。这个队列在每解一个图像，甚至是每个片后都会动态地更新。维护参考帧队列是编解码器十分重要的工作，而本句法元素是维护参考帧队列的重要依据。参考帧队列的复杂的维护机制是 H.264 重要也是很有特色的组成部分

**num\_ref\_idx\_l1\_active\_minus1** 与上一个句法元素的语义一致，只是本句法元素用于 list 1，而上一句法元素用于 list0

**weighted\_pred\_flag** 用以指明是否允许 P 和 S P 片的加权预测，如果允许，在片头会出现用以计算加权预测的句法元素。

**weighted\_bipred\_flag** 用以指明是否允许 B 片的加权预测，本句法元素等于 0 时表示使用默认加权预测模式，等于 1 时表示使用显式加权预测模式，等于 2 时表示使用隐式加权预测模式。

**pic\_init\_qp\_minus26** 加 26 后用以指明亮度分量的量化参数的初始值。在 H.264 中，量化参数分三个级别给出：图像参数集、片头、宏块。在图像参数集给出的是一个初始值。

**pic\_init\_qs\_minus26** 与上一个句法元素语义一致，只是用于 SP 和 SI

**chroma\_qp\_index\_offset** 色度分量的量化参数是根据亮度分量的量化参数计算出来的，本句法元素用以指明计算时用到的参数。

**deblocking\_filter\_control\_present\_flag** 编码器可以通过句法元素显式地控制去块滤波的强度，本句法元素指明是在片头是否会有句法元素传递这个控制信息。如果本句法元素等于 0，那些用于传递滤波强度的句法元素不会出现，解码器将独立地计算出滤波强度。

**constrained\_intra\_pred\_flag** 在 P 和 B 片中，帧内编码的宏块的邻近宏块可能是采用的帧间编码。当本句法元素等于 1 时，表示帧内编码的宏块不能用帧间编码的宏块的像素作为自己的预测，即帧内编码的宏块只能用邻近帧内编码的宏块的像素作为自己的预测；而本句法元素等于 0 时，表示不存在这种限制。

**redundant\_pic\_cnt\_present\_flag** 指明是否会出现 redundant\_pic\_cnt 句法元素。

### 7.3.4 片头语义

**first\_mb\_in\_slice** 片中的第一个宏块的地址，片通过这个句法元素来标定它自己的地址。

要注意的是在帧场自适应模式下，宏块都是成对出现，这时本句法元素表示的是第几个宏块对，对应的第一个宏块的真实地址应该是

$$2 * \text{first\_mb\_in\_slice}$$

**slice\_type** 指明片的类型，具体语义见表7.21。

表 7.21 slice\_type 语义

slice_type	Name of slice_type
0	P (P slice)
1	B (B slice)
2	I (I slice)
3	SP (SP slice)
4	SI (SI slice)
5	P (P slice)
6	B (B slice)
7	I (I slice)
8	SP (SP slice)
9	SI (SI slice)

IDR 图像时, slice\_type 等于 2, 4, 7, 9。

**pic\_parameter\_set\_id** 图像参数集的索引号. 范围 0 到 255。

**frame\_num** 每个参考帧都有一个依次连续的 frame\_num 作为它们的标识,这指明了各图像的解码顺序。但事实上我们在表 中可以看到, frame\_num 的出现没有 if 语句限定条件, 这表明非参考帧的片头也会出现 frame\_num。只是当该个图像是参考帧时, 它所携带的这个句法元素在解码时才有意义。如表 7.21 所示例子:

表 7.21 某个序列的 frame\_num 值  
(该序列中 B 帧一律不作为参考帧, 而 P 帧一律作为参考帧。)

图像序号	图像类型	是否用作参考	frame_num
1	I	是	0
2	P	是	1
3	B	否	2
4	P	是	2
5	B	否	3
6	P	是	3
7	B	否	4
8	P	是	4
...	...	...	...

H.264 对 frame\_num 的值作了如下规定: 当参数集中的句法元素 gaps\_in\_frame\_num\_value\_allowed\_flag 不为 1 时, 每个图像的 frame\_num 值是它前一个参考帧的 frame\_num 值增加 1。这句话包含有两层意思:

1) 当 gaps\_in\_frame\_num\_value\_allowed\_flag 不为 1, 即 frame\_num 连续的情况下, 每个图像的 frame\_num 由前一个参考帧图像对应的值加 1, 着重点是“前一个参考帧”。在表 7.21 中第 3 个图像是 B 帧, 按照定义, 它的 frame\_num 值应是前一个参考帧, 即第 2 个图像对应的值加 1, 即为 2; 第 4 个图像是 P 帧, 由于该序列 B 帧都不作为参考帧, 所以对于该图像来说, 定义中所谓的“前一个参

考帧”，仍旧是指的第 2 个图像，所以对于第 4 个图像来说，它的 `frame_num` 的取值和第 3 个图像一样，也为 2。相同的情况也发生在第 6 和第 8 帧上。

前面我们曾经提到，对于非参考帧来说，它的 `frame_num` 值在解码过程中是没有意义的，因为 `frame_num` 值是参考帧特有的，它的主要作用是在该图像被其他图像引用作运动补偿的参考时提供一个标识。但 H.264 并没有在非参考帧图像中取消这一句法元素，原因是在 POC 的第二种和第三种解码方法中可以通过非参考帧的 `frame_num` 值计算出他们的 POC 值，在第八章中会详细讲述这个问题。

2) 当 `gaps_in_frame_num_value_allowed_flag` 等于 1，前文已经提到，这时若网络阻塞，编码器可以将编码后的若干图像丢弃，而不用另行通知解码器。在这种情况下，解码器必须有机制将缺失的 `frame_num` 及所对应的图像填补，否则后续图像若将运动矢量指向缺失的图像将会产生解码错误。

**field\_pic\_flag** 这是在片层标识图像编码模式的唯一一个句法元素。所谓的编码模式是指的帧编码、场编码、帧场自适应编码。当这个句法元素取值为 1 时 属于场编码；0 时为非场编码。

序列参数集中的句法元素 `frame_mbs_only_flag` 和 `mb_adaptive_frame_field_flag` 再加上本句法元素共同决定图像的编码模式，如图 7.7 所示。

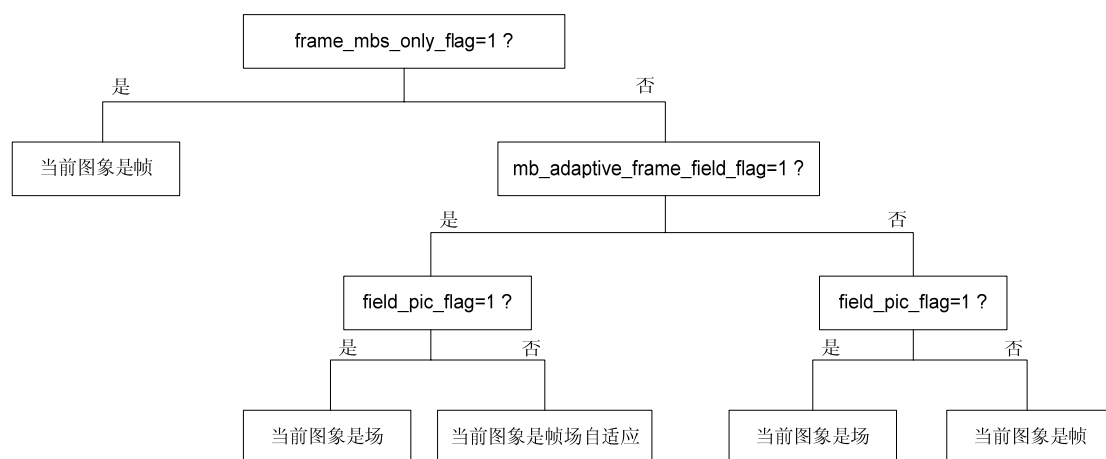


图 7.7 图像编码模式的判断流程

在序列参数集中我们已经能够计算出图像的高和宽的大小，但曾强调那里的高是指的该序列中图像的帧的高度，而一个实际的图像可能是帧也可能是场，对于图像的实际高度，应进一步作如下处理：

$$\text{PicHeightInMbs} = \text{FrameHeightInMbs} / (1 + \text{field\_pic\_flag})$$

从而我们可以得到在解码器端所用到的其他与图像大小有关的变量：

$$\text{PicHeightInSamplesL} = \text{PicHeightInMbs} * 16$$

$$\text{PicHeightInSamplesC} = \text{PicHeightInMbs} * 8$$

$$\text{PicSizeInMbs} = \text{PicWidthInMbs} * \text{PicHeightInMbs}$$

前文已提到，`frame_num` 是参考帧的标识，但是在解码器中，并不是直接引用的 `frame_num` 值，

而是由 `frame_num` 进一步计算出来的变量 `PicNum`,在第八章会详细讲述由 `frame_num` 映射到 `PicNum` 的算法。这里介绍在该算法中用到的两个变量。

**MaxPicNum:**

表征 `PicNum` 的最大值, `PicNum` 和 `frame_num` 一样, 也是嵌在循环中, 当达到这个最大值时, `PicNum` 将从 0 开始重新计数。

- 如果 `field_pic_flag = 0`, `MaxPicNum = MaxFrameNum`.
- 否则, `MaxPicNum = 2 * MaxFrameNum`.

**CurrPicNum:**

当前图像的 `PicNum` 值, 在计算 `PicNum` 的过程中, 当前图像的 `PicNum` 值是由 `frame_num` 直接算出 (在第八章中会看到, 在解某个图像时, 要将已经解码的各参考帧的 `PicNum` 重新计算一遍, 新的值参考当前图像的 `PicNum` 得来)。

- 如果 `field_pic_flag = 0`, `CurrPicNum = frame_num`.
- 否则, `CurrPicNum = 2 * frame_num + 1`.

`Frame_num` 是对帧编号的, 也就是说如果在场模式下, 同属一个场对的顶场和底场两个图像的 `frame_num` 的值是相同的。在帧或帧场自适应模式下, 就直接将图像的 `frame_num` 赋给 `PicNum`, 而在场模式下, 将 `2 * frame_num` 和 `2 * frame_num + 1` 两个值分别赋给两个场。`2 * frame_num + 1` 这个值永远被赋给当前场, 解码到当前场对的下一个场时, 刚才被赋为 `2 * frame_num + 1` 的场的 `PicNum` 值被重新计算为 `2 * frame_num`, 而将 `2 * frame_num + 1` 赋给新的当前场。

**bottom\_field\_flag** 等于 1 时表示当前图像是属于底场; 等于 0 时表示当前图像是属于顶场。

**idr\_pic\_id** IDR 图像的标识。不同的 IDR 图像有不同的 `idr_pic_id` 值。值得注意的是, IDR 图像有不等价于 I 图像, 只有在作为 IDR 图像的 I 帧才有这个句法元素, 在场模式下, IDR 帧的两个场有相同的 `idr_pic_id` 值。`idr_pic_id` 的取值范围是 `[0, 65535]`, 和 `frame_num` 类似, 当它的值超出这个范围时, 它会以循环的方式重新开始计数。

**pic\_order\_cnt\_lsb** 在 POC 的第一种算法中本句法元素来计算 POC 值, 在 POC 的第一种算法中是显式地传递 POC 的值, 而其他两种算法是通过 `frame_num` 来映射 POC 的值。注意这个句法元素的读取函数是 `u(v)`, 这个 `v` 的来自是图像参数集:

$$v = \log_2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4。$$

**delta\_pic\_order\_cnt\_bottom** 如果是在场模式下, 场对中的两个场都各自被构造为一个图像, 它们有各自的 POC 算法来分别计算两个场的 POC 值, 也就是一个场对拥有一对 POC 值; 而在是帧模式或是帧场自适应模式下, 一个图像只能根据片头的句法元素计算出一个 POC 值。根据 H.264 的规定, 在序列中有可能出现场的情况, 即 `frame_mbs_only_flag` 不为 1 时, 每个帧或帧场自适应的图像在解码完后必须分解为两个场, 以供后续图像中的场作为参考图像。所以当 `frame_mb_only_flag` 不为 1 时, 帧或帧场自适应中包含的两个场也必须有各自的 POC 值。在第八章中我们会看到, 通过本句法元素, 可以在已经解开的帧或帧场自适应图像的 POC 基础上新映射一个 POC 值, 并把它赋给底场。当然, 象句法表指出的那样, 这个句法元素只用在 POC 的第一个算法中。

**delta\_pic\_order\_cnt[ 0 ], delta\_pic\_order\_cnt[ 1 ]:**

前文已经提到，POC 的第二和第三种算法是从 `frame_num` 映射得来，这两个句法元素用于映射算法。`delta_pic_order_cnt[ 0 ]` 用于帧编码方式下的底场和场编码方式的场，`delta_pic_order_cnt[ 1 ]` 用于帧编码方式下的顶场。在第八章会详细讲述 POC 的三种算法。

**redundant\_pic\_cnt** 冗余片的 id 号。

**direct\_spatial\_mv\_pred\_flag** 指出在 B 图像的直接预测的模式下，用时间预测还是用空间预测。1：空间预测；0：时间预测。

**num\_ref\_idx\_active\_override\_flag** 在图像参数集中我们看到已经出现句法元素 `num_ref_idx_l0_active_minus1` 和 `num_ref_idx_l1_active_minus1` 指定当前参考帧队列中实际可用的参考帧的数目。在片头可以重载这对句法元素，以给某特定图像更大的灵活度。这个句法元素就是指明片头是否会重载，如果该句法元素等于 1，下面会出现新的 `num_ref_idx_l0_active_minus1` 和 `num_ref_idx_l1_active_minus1` 值。

**num\_ref\_idx\_l0\_active\_minus1** 、 **num\_ref\_idx\_l1\_active\_minus1**

如上个句法元素中所介绍，这是重载的 `num_ref_idx_l0_active_minus1` 及 `num_ref_idx_l1_active_minus1`

**cabac\_init\_idc** 给出 cabac 初始化时表格的选择，范围 0 到 2。

**slice\_qp\_delta** 指出在用于当前片的所有宏块的量化参数的初始值。 $QP_Y$

$$\text{SliceQP}_Y = 26 + \text{pic\_init\_qp\_minus26} + \text{slice\_qp\_delta}$$

$QP_Y$  的范围是 0 to 51。

我们前文已经提到，H.264 中量化参数是分图像参数集、片头、宏块头三层给出的，前两层各自给出一个偏移值，这个句法元素就是片层的偏移。

**sp\_for\_switch\_flag** 指出 SP 帧中的 p 宏块的解码方式是否是 switching 模式，在第八章有详细的说明。

**slice\_qs\_delta** 与 `slice_qp_delta` 的与语义相似，用在 SI 和 SP 中的

$$QS_Y = 26 + \text{pic\_init\_qs\_minus26} + \text{slice\_qs\_delta}$$

$QS_Y$  值的范围是 0 到 51。

**disable\_deblocking\_filter\_idc** H.264 指定了一套算法可以在解码器端独立地计算图像中各边界的滤波强度进行滤波。除了解码器独立计算之外，编码器也可以传递句法元素来干涉滤波强度，当这个句法元素指定了在块的边界是否要用滤波，同时指明那个块的边界不用块滤波，在第八章会详细讲述。

**slice\_alpha\_c0\_offset\_div2** 给出用于增强  $\alpha$  和  $t_{c0}$  的偏移值

$$\text{FilterOffset}_A = \text{slice\_alpha\_c0\_offset\_div2} \ll 1$$

`slice_alpha_c0_offset_div2` 值的范围是 -6 到+6。



**slice\_beta\_offset\_div2** 给出用于增强  $\beta$  和  $t_{c0}$  的偏移值

$$\text{FilterOffsetB} = \text{slice\_beta\_offset\_div2} \ll 1$$

**slice\_beta\_offset\_div2** 值的范围是 -6 到+6。

**slice\_group\_change\_cycle** 当片组的类型是 3, 4, 5, 由句法元素可获得片组中 映射单元的数目:

$$\text{MapUnitsInSliceGroup0} = \text{Min}(\text{slice\_group\_change\_cycle} * \text{SliceGroupChangeRate}, \text{PicSizeInMapUnits})$$

**slice\_group\_change\_cycle** 由  $\text{Ceil}(\text{Log2}(\text{PicSizeInMapUnits} \div \text{SliceGroupChangeRate} + 1))$  位比特表示

**slice\_group\_change\_cycle** 值的范围是 0 到  $\text{Ceil}(\text{PicSizeInMapUnits} \div \text{SliceGroupChangeRate})$ 。

### 7.3.5 参考图像序列重排序的语义

每一个使用帧间预测的图像都会引用前面已解码的图像作为参考帧。如前文所述, 编码器给每个参考帧都会分配一个唯一性的标识, 即句法元素 **frame\_num**。但是, 当编码器要指定当前图像的参考图像时, 并不是直接指定该图像的 **frame\_num** 值, 而是使用通过下面步骤最终得出的 **ref\_id** 号:

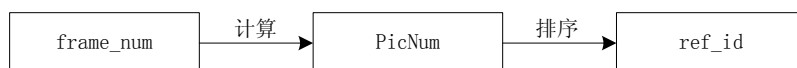


图 7.8 由 **frame\_num** 到 **ref\_id**

其中, 从 **frame\_num** 到变换到变量 **PicNum** 主要是考虑到场模式的需要, 当序列中允许出现场时, 每个非场的图像(帧或 帧场自适应)解码后必须分解为一个场对, 从而需要为它们分解后的两个场各自指定一个标识; 进一步从 **PicNum** 到 **ref\_id** 是为了能节省码流, 因为 **PicNum** 的值通常都比较大, 而在帧间预测时, 需要为每个运动矢量都指明相对应的参考帧的标识, 如果这个标识选用 **PicNum** 开销就会比较大, 所以 H.264 又将 **PicNum** 映射为一个更小的变量 **ref\_id**。在编码器和解码器都同步地维护一个参考帧队列, 每解码一个片就将该队列刷新一次, 把各图像按照特定的规则进行排序, 排序后各图像在队列中的序号就是该图像的 **ref\_id** 值, 在下文中我们可以看到在宏块层表示参考图像的标识就是 **ref\_id**, 在第八章我们会详细讲述队列的初始化、排序等维护算法。本节和下节介绍的是在维护队列时两个重要操作: 重排序(reordering)和标记(marking)所用到的句法元素。

说明: 句法元素的后缀名带有 L0 指的是第一个参数列表; 句法元素的后缀名带有 L1 指的是第二个参数列表(用在 B 帧预测中)。

**ref\_pic\_list\_reordering\_flag\_l0** 指明是否进行重排序操作, 这个句法元素等于 1 时表明紧跟着会有一系列句法元素用于参考帧队列的重排序。

**re\_pic\_list\_reordering\_flag\_l1** 与上个句法元素的语义相同, 只是本句法元素用于参考帧队列 L1。

**reordering\_of\_pic\_nums\_idc** 指明执行哪种重排序操作, 具体语义见表 7.22。

表 7.22 重排序操作

reordering_of_pic_nums_idc	操作
0	短期参考帧重排序，abs_diff_pic_num_minus1会出现在码流中，从当前图像的PicNum减去 (abs_diff_pic_num_minus1 + 1) 后指明需要重排序的图像。
1	短期参考帧重排序，abs_diff_pic_num_minus1会出现在码流中，从当前图像的PicNum加上 (abs_diff_pic_num_minus1 + 1) 后指明需要重排序的图像。
2	长期参考帧重排序，long_term_pic_num会出现在码流中，指明需要重排序的图像。
3	结束循环，退出重排序操作。

**abs\_diff\_pic\_num\_minus1** 在对短期参考帧重排序时指明重排序图像与当前的差。见表 7.22。

**long\_term\_pic\_num** 在对长期参考帧重排序时指明重排序图像。见表 7.22。

从上文可以看到，每组 reordering\_of\_pic\_nums\_idc、abs\_diff\_pic\_num\_minus1 或 reordering\_of\_pic\_nums\_idc、long\_term\_pic\_num 只能对一个图像操作，而通常情况下都需要对一组图像重排序，所以在码流中一般会有个循环，反复出现这些重排序的句法元素，如图，循环直到 reordering\_of\_pic\_nums\_id 等于 3 结束，我们在表 7.22 中也可以看到这种情况。

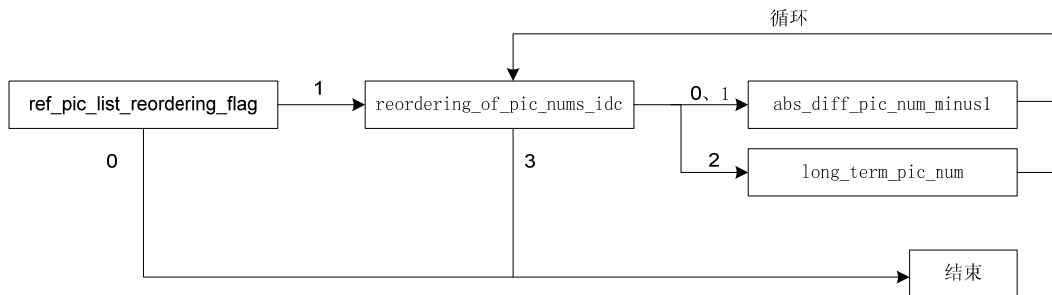


图 7.8 重排序流程

### 7.3.6 加权预测的语义

**luma\_log2\_weight\_denom** 给出参考帧列表中参考图像所有亮度的加权系数，是个初始值 luma\_log2\_weight\_denom 值的范围是 0 to 7。

**chroma\_log2\_weight\_denom** 给出参考帧列表中参考图像所有色度的加权系数，是个初始值 chroma\_log2\_weight\_denom 值的范围是 0 to 7。

**luma\_weight\_l0\_flag** 等于 1 时，指的是在参考序列 0 中的亮度的加权系数存在；等于 0 时，在参考序列 0 中的亮度的加权系数不存在。

**luma\_weight\_l0[i]** 用参考序列 0 预测亮度值时，所用的加权系数。如果 luma\_weight\_l0\_flag is = 0,  $luma\_weight\_l0[i] = 2^{luma\_log2\_weight\_denom}$ 。

**luma\_offset\_l0[i]** 用参考序列 0 预测亮度值时,所用的加权系数的额外的偏移。**luma\_offset\_l0[i]** 值的范围-128 to 127。如果 **luma\_weight\_l0\_flag** is = 0, **luma\_offset\_l0[i] = 0** 。

**chroma\_weight\_l0\_flag**, **chroma\_weight\_l0[i][j]** , **chroma\_offset\_l0[i][j]** 与上述三个类似, 不同的是这三个句法元素是用在色度预测。

**luma\_weight\_l1\_flag**, **luma\_weight\_l1**, **luma\_offset\_l1**, **chroma\_weight\_l1\_flag**, **chroma\_weight\_l1**, **chroma\_offset\_l1**。

与 **luma\_weight\_l0\_flag**, **luma\_weight\_l0**, **luma\_offset\_l0**, **chroma\_weight\_l0\_flag**, **chroma\_weight\_l0**, **chroma\_offset\_l0** 的语义相似, 不同点是这几个句法元素是用在参考序列 1 中。

**7.3.7 参考图像序列标记 (marking)操作的语义**

前文介绍的重排序 (reordering) 操作是对参考帧队列重新排序, 而**标记 (marking) 操作负责将参考图像移入或移出参考帧队列。**

**no\_output\_of\_prior\_pics\_flag** 仅在当前图像是 IDR 图像时出现这个句法元素, 指明是否要将前面已解码的图像全部输出。

**long\_term\_reference\_flag** 与上个图像一样, 仅在当前图像是 IDR 图像时出现这一句法元素。这个句法元素指明是否使用长期参考这个机制。**如果取值为 1, 表明使用长期参考, 并且每个 IDR 图像被解码后自动成为长期参考帧, 否则 (取值为 0), IDR 图像被解码后自动成为短期参考帧。**

**adaptive\_ref\_pic\_marking\_mode\_flag** 指明标记 (marking) 操作的模式, 具体语义见表 7.23。

表 7.23 标记 (marding) 模式

<b>adaptive_ref_pic_marking_mode_flag</b>	<b>标记 (marking) 模式</b>
0	先入先出 (FIFO): 使用滑动窗的机制, 先入先出, 在这种模式下没有办法对长期参考帧进行操作。
1	自适应标记 (marking): 后续码流中会有一系列句法元素显式指明操作的步骤。自适应是指编码器可根据情况随机灵活地作出决策。

**memory\_management\_control\_operation** 在自适应标记 (marking) 模式中, 指明本次操作的具体内容, 具体语义见表 7.24。

表 7.24 标记（marking）操作

memory_management_control_operation	标记（marking）操作
0	结束循环，退出标记（marking）操作。
1	将一个短期参考图像标记为非参考图像，也即将一个短期参考图像移出参考帧队列。
2	将一个长期参考图像标记为非参考图像，也即将一个长期参考图像移出参考帧队列。
3	将一个短期参考图像转为长期参考图像。
4	指明长期参考帧的最大数目。
5	清空参考帧队列，将所有参考图像移出参考帧队列，并禁用长期参考机制
6	将当前图像存为一个长期参考帧。

**difference\_of\_pic\_nums\_minus1** 当 memory\_management\_control\_operation 等于 3 或 1 时，由 这个句法元素可以计算得到需要操作的图像在短期参考队列中的序号。参考帧队列中必须存在这个图像。

**long\_term\_pic\_num** 当 memory\_management\_control\_operation 等于 2 时， 从此句法元素得到所要操作的长期参考图像的序号。

**long\_term\_frame\_idx** 当 memory\_management\_control\_operation 等于 3 或 6 ， 分配一个长期参考帧的序号给一个图像。

**max\_long\_term\_frame\_idx\_plus1** 此句法元素减 1， 指明长期参考队列的最大数目。  
max\_long\_term\_frame\_idx\_plus1 值的范围 0 to num\_ref\_frames。

### 7.3.8 片数据的语义

**cabac\_alignment\_one\_bit** 当熵编码模式是 CABAC 时,此时要求数据字节对齐,即数据从下一个字节的第一个比特开始,如果还没有字节对齐将出现若干个 **cabac\_alignment\_one\_bit** 作为填充。

**mb\_skip\_run** 当图像采用帧间预测编码时，H.264 允许在图像平坦的区域使用“跳跃”块，“跳跃”块本身不携带任何数据，解码器通过周围已重建的宏块的数据来恢复“跳跃”块。在表 7.3 我们可以看到，当熵编码为 CAVLC 或 CABAC 时，“跳跃”块的表示方法不同。当 entropy\_coding\_mode\_flag 为 1，即熵编码为 CABAC 时，是每个“跳跃”块都会有句法元素 mb\_skip\_flag 指明，而 entropy\_coding\_mode\_flag 等于 0，即熵编码为 CAVLC 时，用一种行程的方法给出紧连着的“跳跃”块的数目，即句法元素 mb\_skip\_run。mb\_skip\_run 值的范围 0 to PicSizeInMbs – CurrMbAddr 。

**mb\_skip\_flag** 见上一条，指明当前宏块是否是跳跃编码模式的宏块。

**mb\_field\_decoding\_flag** 在帧场自适应图像中，指明当前宏块所属的宏块对是帧模式还是场模式。0 帧模式；1 场模式。如果一个宏块对的两个宏块句法结构中都没有出现这个句法元素，即它们都是“跳跃”块时，本句法元素由以下决定：

- 如果这个宏块对与相邻的、左边的宏块对属于同一个片时，这个宏块对的 mb\_field\_decoding\_flag 的值等于左边的宏块对的 mb\_field\_decoding\_flag 的值。
- 否则，这个宏块对的 mb\_field\_decoding\_flag 的值等于上边同属于一个片的宏块对的 mb\_field\_decoding\_flag 的值。

- 如果这个宏块对既没有相邻的、上边同属于一个片的宏块对；也没有相邻的、左边同属于一个片的宏块对，这个宏块对的 `mb_field_decoding_flag` 的值等于 0，即帧模式。  
`end_of_slice_flag` 指明是否到了片的结尾。

### 7.3.9 宏块层的语义

**mb\_type** 指明当前宏块的类型。H.264规定，不同的片中允许出现的宏块类型也不同。下表指明在各种片类型中允许出现的宏块种类。

表 7.25 各种片中允许出现的宏块类型

片类型	允许出现的宏块种类
I (slice)	I 宏块
P (slice)	P 宏块、 I 宏块
B (slice)	B 宏块、 I 宏块
SI (slice)	SI 宏块、 I 宏块
SP (slice)	P 宏块、 I 宏块

可以看到，I 片中只允许出现 I 宏块，而 P 片中即可以出现 P 宏块也可以出现 I 宏块，也就是说，在帧间预测的图像中也可以包括帧内预测的图像。其它片也有类似情况。

每一种宏块包含许多的类型。比起以往的视频编码标准，H.264 定义了更多的宏块的类型。

在帧间预测模式下，宏块可以有七种运动矢量的划分方法。

在帧内预测模式下，可以是帧内 16x16 预测，这时可以宏块有四种预测方法，即四种类型；也可以是 4x4 预测，这时每个 4x4 块可以有九种预测方法，整个宏块共有 144 种类型。

**mb\_type** 并不能描述以上所有有关宏块类型的信息。事实上可以体会到，**mb\_tye** 是出现在宏块层的第一个句法元素，它描述跟整个宏块有关的基本的类型信息。在不同的片中 **mb\_type** 的定义是不同的，下面我们分别讨论 I、P、B 片中这个句法元素的意义。

a) I 片中的 **mb\_type**,具体语义见表

表 7.26 I 片中的 mb\_type

mb_type	类型名称	预测方式	帧内16x16的预测模式	CodedBlockPatternChroma	CodedBlockPatternLuma
0	I_4x4	Intra_4x4	无	无	无
1	I_16x16_0_0_0	Intra_16x16	0	0	0
2	I_16x16_1_0_0	Intra_16x16	1	0	0
3	I_16x16_2_0_0	Intra_16x16	2	0	0
4	I_16x16_3_0_0	Intra_16x16	3	0	0
5	I_16x16_0_1_0	Intra_16x16	0	1	0
6	I_16x16_1_1_0	Intra_16x16	1	1	0
7	I_16x16_2_1_0	Intra_16x16	2	1	0
8	I_16x16_3_1_0	Intra_16x16	3	1	0
9	I_16x16_0_2_0	Intra_16x16	0	2	0
10	I_16x16_1_2_0	Intra_16x16	1	2	0
11	I_16x16_2_2_0	Intra_16x16	2	2	0
12	I_16x16_3_2_0	Intra_16x16	3	2	0
13	I_16x16_0_0_1	Intra_16x16	0	0	15
14	I_16x16_1_0_1	Intra_16x16	1	0	15
15	I_16x16_2_0_1	Intra_16x16	2	0	15
16	I_16x16_3_0_1	Intra_16x16	3	0	15
17	I_16x16_0_1_1	Intra_16x16	0	1	15
18	I_16x16_1_1_1	Intra_16x16	1	1	15
19	I_16x16_2_1_1	Intra_16x16	2	1	15
20	I_16x16_3_1_1	Intra_16x16	3	1	15
21	I_16x16_0_2_1	Intra_16x16	0	2	15
22	I_16x16_1_2_1	Intra_16x16	1	2	15
23	I_16x16_2_2_1	Intra_16x16	2	2	15
24	I_16x16_3_2_1	Intra_16x16	3	2	15
25	I_PCM	无	无	无	无

表中，Intra\_4x4 表示使用帧内 4x4 预测，Intra\_16x16 表示使用帧内 16x16 预测。当使用帧内 16x16 时，类型名称由了如下的结构组成：

$$I_{16x16\_x\_y\_z}$$

其中, x 对应于表中“帧内 16x16 的预测模式”字段的值, y 对应于表中“色度 CBP”字段的值, z 对应于表中“亮度 CBP”的值。

- 帧内 16x16 的预测模式: 当使用帧内 16x16 预测时, 指定使用何种预测方式, 帧内 16x16 共有四种预测模式, 第八章中会详细介绍这些预测模式的算法。
- CodedBlockPatternLuma: 指定当前宏块色度分量的 CBP, CBP (CodedBlockPattern) 是指子宏块残差的编码方案。该变量详细语义见 coded\_block\_pattern 条目。
- 亮度 CBP: 指定当前宏块亮度分量的 CBP, 详细语义见 coded\_block\_pattern 条目。

我们看到, 帧内 16x16 宏块类型的 mb\_type 语义原比其它宏块类型的复杂, 这是因为当使用帧内 16x16 时, 整个宏块是一个统一的整体, 宏块中各子宏块、4x4 小块的预测模式信息都是相同的, 所以可以把这些信息放入 mb\_type, 以减少码流。其它宏块类型的这些信息必须在各子块中另外用句法元素指明。

b) P 片中的 mb\_type, 具体语义见表 7.26。

表 7.26 P 片中的 mb\_type

mb_type	类型名称	宏块分区数目	预测模式 (mb_type,0)	预测模式 (mb_type,1)	宏块分区宽度 (mb_type)	宏块分区高度 (mb_type)
0	P_L0_16x16	1	Pred_L0	无	16	16
1	P_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
2	P_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
3	P_8x8	4	无	无	8	8
4	P_8x8ref0	4	无	无	8	8
无	P_Skip	1	Pred_L0	无	16	16

在表 7.26 中, Pred\_L0 表示用 L0, 即前向预测。如果当前宏块的 mb\_type 等于 0 到 4, mb\_type 的含义见 表 7.26; 当 mb\_type 等于 5 到 30 时, mb\_type 的含义见 表 7.25, 用 mb\_type-5 所得到的值来进行查找。预测模式 (mb\_type,n) 预测模式是 mb\_type 的函数, n 是宏块的第 n 个分区。

c) B 片中的 mb\_type, 具体语义见表 7.27。

如果当前宏块是属于 B 片且 mb\_type 等于 0 到 22, mb\_type 的含义见 表 7-11; 当 mb\_type 等于 23 到 48 时, mb\_type 的含义见 表 7-8, 用 mb\_type-23 所得到的值来进行查找。

表 7.27 B 片中的 mb\_type



mb_type	类型名称	宏块分区数目 (mb_type)	预测模式 (mb_type, 0)	预测模式 (mb_type, 1)	宏块分区宽度 (mb_type)	宏块分区高度 (mb_type)
0	B_Direct_16x16	无	Direct	无	8	8
1	B_L0_16x16	1	Pred_L0	无	16	16
2	B_L1_16x16	1	Pred_L1	无	16	16
3	B_Bi_16x16	1	BiPred	无	16	16
4	B_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
5	B_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
6	B_L1_L1_16x8	2	Pred_L1	Pred_L1	16	8
7	B_L1_L1_8x16	2	Pred_L1	Pred_L1	8	16
8	B_L0_L1_16x8	2	Pred_L0	Pred_L1	16	8
9	B_L0_L1_8x16	2	Pred_L0	Pred_L1	8	16
10	B_L1_L0_16x8	2	Pred_L1	Pred_L0	16	8
11	B_L1_L0_8x16	2	Pred_L1	Pred_L0	8	16
12	B_L0_Bi_16x8	2	Pred_L0	BiPred	16	8
13	B_L0_Bi_8x16	2	Pred_L0	BiPred	8	16
14	B_L1_Bi_16x8	2	Pred_L1	BiPred	16	8
15	B_L1_Bi_8x16	2	Pred_L1	BiPred	8	16
16	B_Bi_L0_16x8	2	BiPred	Pred_L0	16	8
17	B_Bi_L0_8x16	2	BiPred	Pred_L0	8	16
18	B_Bi_L1_16x8	2	BiPred	Pred_L1	16	8
19	B_Bi_L1_8x16	2	BiPred	Pred_L1	8	16
20	B_Bi_Bi_16x8	2	BiPred	BiPred	16	8
21	B_Bi_Bi_8x16	2	BiPred	BiPred	8	16
22	B_8x8	4	无	无	8	8
无	B_Skip	无	Direct	无	8	8

表中，Pred\_L0 表示使用 L0，即前向预测，Pred\_L1 表示使用 L1，即后向预测，Bipred 表示双向预测，Direct 表示直接预测模式。预测模式 (mb\_type,n) 预测模式是 mb\_type 的函数，n 是宏块的第 n 个分区。

**pcm\_alignment\_zero\_bit** 等于 0。

**pcm\_byte[ i ]** 像素值。前 256 pcm\_byte[ i ] 的值代表亮度像素的值，下一个

$(256 * (\text{ChromaFormatFactor} - 1)) / 2$  个 `pcm_byte[i]` 的值代表 Cb 分量的值。最后一个  $(256 * (\text{ChromaFormatFactor} - 1)) / 2$  个 `pcm_byte[i]` 的值代表 Cr 分量的值。

**coded\_block\_pattern** 即 CBP，指亮度和色度分量的各小块的残差的编码方案，所谓编码方案有以下几种：

- a) 所有残差（包括 DC、AC）都编码。
- b) 只对 DC 系数编码。
- c) 所有残差（包括 DC、AC）都不编码。

这个句法元素同时隐含了一个宏块中亮度、色度分量的 CBP，所以第一步必须先分别解算出各分量各自 CBP 的值。其中，两个色度分量的 CBP 是相同的。变量 `CodedBlockPatternLuma` 是亮度分量的 CBP，变量 `CodedBlockPatternChroma` 是色度分量的 CBP：

对于非 `Intra_16x16` 的宏块类型：

```
CodedBlockPatternLuma = coded_block_pattern % 16
CodedBlockPatternChroma = coded_block_pattern / 16
```

对于 `Intra_16x16` 宏块类型，`CodedBlockPatternLuma` 和 `CodedBlockPatternChroma` 的值不是由本句法元素给出，而是通过 `mb_type` 得到。

- `CodedBlockPatternLuma`：是一个 16 位的变量，其中只有最低四位有定义。由于非 `Intra_16x16` 的宏块不单独编码 DC 系数，所以这个变量只指明两种编码方案：残差全部编码或全部不编码。变量的最低位比特从最低位开始，每一位对应一个子宏块，该位等于 1 时表明对应子宏块残差系数被传送；该位等于 0 时表明对应子宏块残差全部不被传送，解码器把这些残差系数赋为 0。
- `CodedBlockPatternChroma`：当值为 0、1、2 时有定义，见表 7.28。

表 7.28 `CodedBlockPatternChroma` 的定义

<code>CodedBlockPatternChroma</code>	定义
0	所有残差都不被传送，解码器把所有残差系数赋为 0。
1	只有 DC 系数被传送，解码器把所有 AC 系数赋为 0。
2	所有残差系数（包括 DC、AC）都被传送。解码器用接收到的残差系数重建图像。

**mb\_qp\_delta** 在宏块层中的量化参数的偏移值。`mb_qp_delta` 值的范围是 -26 to +25。量化参数是在图像参数集、片头、宏块分三层给出的，最终用于解码的量化参数由以下公式得到：

$$QP_Y = (QP_{Y,PREV} + mb\_qp\_delta + 52) \% 52 \quad (7-23)$$

$QP_{Y,PREV}$  是当前宏块按照解码顺序的前一个宏块的量化参数，我们可以看到，`mb_qp_delta` 所指示的偏移是前后两个宏块之间的偏移。而对于片中第一个宏块的  $QP_{Y,PREV}$  是由 7-16 式给出

$$QP_{Y,PREV} = 26 + pic\_init\_qp\_minus26 + slice\_qp\_delta$$

### 7.3.10 宏块预测的语义

`prev_intra4x4_pred_mode_flag[ luma4x4BlkIdx ] rem_intra4x4_pred_mode[ luma4x4BlkIdx ]`

帧内预测的模式也是需要预测的， `prev_intra4x4_pred_mode_flag` 用来指明帧内预测时，亮度分量的预测模式的预测值是否就是真实预测模式，如果是，就不需另外再传预测模式。如果不是，就由 `rem_intra4x4_pred_mode` 指定真实预测模式。

**`intra_chroma_pred_mode`** 在帧内预测时指定色度的预测模式，具体语义见表 7.29。

表 7.29 `intra_chroma_pred_mode` 的语义

<code>intra_chroma_pred_mode</code>	预测模式
0	DC
1	Horizontal
2	Vertical
3	Plane

**`ref_idx_l0[ mbPartIdx ]`** 用参考帧队列 L0 进行预测，即前向预测时，参考图像在参考帧队列中的序号。其中 `mbPartIdx` 是宏块分区的序号。

如果当前宏块是非场宏块，则 `ref_idx_l0[ mbPartIdx ]` 值的范围是 0 到 `num_ref_idx_l0_active_minus1`。

否则，如果当前宏块是场宏块，（宏块所在图像是场，当图像是帧场自适应时当前宏块处于场编码的宏块对），`ref_idx_l0[ mbPartIdx ]` 值的范围是 0 到  $2 * \text{num\_ref\_idx\_l0\_active\_minus1} + 1$ ，如前所述，此时参考帧队列的帧都将拆成场，故参考队列长度加倍。

**`ref_idx_l1[ mbPartIdx ]`** 语义同上，只是这个句法元素用于参考帧队列 L1，即后向预测。

**`mvd_l0[ mbPartIdx ][ 0 ][ compIdx ]`** 运动矢量的预测值和实际值之间的差。`mbPartIdx` 是宏块分区的序号。`CompIdx = 0` 时水平运动矢量； `CompIdx = 1` 垂直运动矢量。

**`mvd_l1[ mbPartIdx ][ 0 ][ compIdx ]`** 语义同上，只是这个句法元素用于 List1，即后向预测。

### 7.3.11 子宏块预测的语义

**`sub_mb_type[ mbPartIdx ]`** 指明子宏块的预测类型，在不同的宏块类型中这个句法元素的语义不一样。

a) P 宏块

表 7.30 P 宏块中的子宏块类型

sub_mb_type[ mbPartIdx ]	sub_mb_type[ mbPartIdx ]	NumSubMbPart ( sub_mb_type[ mbPartIdx ] )	SubMbPredMode ( sub_mb_type[ mbPartIdx ] )	SubMbPartWidth ( sub_mb_type[ mbPartIdx ] )	SubMbPartHeight ( sub_mb_type[ mbPartIdx ] )
0	P_L0_8x8	1	Pred_L0	8	8
1	P_L0_8x4	2	Pred_L0	8	4
2	P_L0_4x8	2	Pred_L0	4	8
3	P_L0_4x4	4	Pred_L0	4	4

## b) B 宏块

表 7.31 B 宏块中的子宏块类型

sub_mb_type[ mbPartIdx ]	Name of sub_mb_type[ mbPartIdx ]	NumSubMbPart ( sub_mb_type[ mbPartIdx ] )	SubMbPredMode ( sub_mb_type[ mbPartIdx ] )	SubMbPartWidth ( sub_mb_type[ mbPartIdx ] )	SubMbPartHeight ( sub_mb_type[ mbPartIdx ] )
0	B_Direct_8x8	na	Direct	4	4
1	B_L0_8x8	1	Pred_L0	8	8
2	B_L1_8x8	1	Pred_L1	8	8
3	B_Bi_8x8	1	BiPred	8	8
4	B_L0_8x4	2	Pred_L0	8	4
5	B_L0_4x8	2	Pred_L0	4	8
6	B_L1_8x4	2	Pred_L1	8	4
7	B_L1_4x8	2	Pred_L1	4	8
8	B_Bi_8x4	2	BiPred	8	4
9	B_Bi_4x8	2	BiPred	4	8
10	B_L0_4x4	4	Pred_L0	4	4
11	B_L1_4x4	4	Pred_L1	4	4
12	B_Bi_4x4	4	BiPred	4	4

ref\_idx\_l0[ mbPartIdx ] 与 3.11 中定义的 ref\_idx\_l0 相同.

**ref\_idx\_11**[ mbPartIdx ] 与 3.11 中定义的 ref\_idx\_11 相同。

**mvd\_l0**[ mbPartIdx ][ subMbPartIdx ][ compIdx ] 与 7.3.11 中定义 mvd\_l0 相同,除了 subMbPartIdx: 子宏块中的索引值。

**mvd\_11**[ mbPartIdx ][ subMbPartIdx ][ compIdx ]同上, list1

### 7.3.12 用 CAVLC 方式编码的残差数据的语义

**coeff\_token** 指明了非零系数的个数, 拖尾系数的个数。

**trailing\_ones\_sign\_flag** 拖尾系数的符号

- 如果trailing\_ones\_sign\_flag = 0, 相应的拖尾系数是+1。
- 否则, trailing\_ones\_sign\_flag =1, 相应的拖尾系数是-1。

-

**level\_prefix** and **level\_suffix** 非零系数值的前缀和后缀。

**total\_zeros** 系数中 0 的总个数。

**run\_before** 在非零系数之前连续零的个数。

### 7.3.13 用 CABAC 方式编码的残差数据的语义

**coded\_block\_flag** 指出当前块是否包含非零系数。

如果 coded\_block\_flag= 0, 这个块不包含非零系数。

如果 coded\_block\_flag = 1, 这个块包含非零系数。

**significant\_coeff\_flag**[ i ] 指出在位置为 i 处的变换系数是否为零。

如果 significant\_coeff\_flag[ i ] = 0, 在位置为 i 处的变换系数为零。

否则, significant\_coeff\_flag[ i ] =1, 在位置为 i 处的变换系数不为零。

**last\_significant\_coeff\_flag**[ i ] 表示当前位置 i 处的变换系数是否为块中最后一个非零系数。

如果 last\_significant\_coeff\_flag[ i ] =1, 这个块中随后的系数都为零。

否则, 这个块中随后的系数中还有其它的非零系数。

**coeff\_abs\_level\_minus1**[ i ]系数的绝对值减 1。

**coeff\_sign\_flag**[ i ] 系数的符号位。

- coeff\_sign\_flag = 0, 正数。
- coeff\_sign\_flag=1, 负数。

## 7.4 总结

H.264 的句法是经过精心设计的, 构成句法的各句法元素既相互依赖而又相互独立。依赖是为了减少冗余信息, 提高编码效率; 而独立是为了使通信更加鲁棒, 在错误发生时限制错误的扩散。但是, 依赖和独立又是矛盾的, 往往要在它们之间作取舍。在实现 H.264 的编解码器时, 同样有速度和健壮性这样一组矛盾的关系, 既要在现有硬件平台上提高编解码的速度, 又要作充分的安全检查、建立异常处理机制, 如果能详细考察句法元素间的依赖与独立关系, 充分利用 H.264 各句法元

素间本身的内在联系与安全机制,便能够设计出即快速, 又安全、健壮的编解码算法。

## 参考文献

1. “Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC,”in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050,2003.
2. T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, “Overview of the H.264/AVC Video Coding Standard,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 560–576, July 2003.
3. S. Wenger, “H.264/AVC over IP,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 645–656, July 2003.
4. T. Stockhammer, M. M. Hannuksela, and T. Wiegand, “H.264/AVC in wireless environments,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 657–673, July 2003.
5. T. Wedi, “Motion compensation in H.264/AVC,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 577–586, July 2003.
6. M. Flierl and B. Girod, “Generalized B pictures and the draft JVT/H.264 video compression standard,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 587–597, July 2003.
7. T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, “Rate-constrained coder control and comparison of video coding standards,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 688–703, July 2003.
8. Schafer Ralf, Wiegand Thomas, Schwarz Heiko. “The emerging H.264/AVC Standard EBU Technical Review”, Jan.2003
9. M. Karczewicz and R. Kurçeren, “The SP and SI frames design for H.264/AVC,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 637–644, July 2003.
10. H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, “Low-Complexity transform and quantization in H.264/AVC,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 598–603, July 2003.
11. J. Ribas-Corbera, P. A. Chou, and S. Regunathan, “A generalized hypothetical reference decoder for H.264/AVC,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 674–687, July 2003.
12. Mathias Wien, “Variable Block-Size Transforms for H.264/AVC” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 604–613, July 2003.
13. Michael Horowitz, Anthony Joch, Faouzi Kossentini, “H.264/AVC Baseline Profile Decoder Complexity Analysis” IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 704–716, July 2003.

## 第 8 章 H.264/AVC 解码器的原理和实现

### 8.1 解码器原理

如前所述，H.264 包含 VCL（视频编码层）和 NAL（网络提取层）。VCL 包括核心压缩引擎和块、宏块和片的语法级别定义，它的设计目标是尽可能地独立于网络得进行高效地编解码；而 NAL 则负责将 VCL 产生的比特字符串适配到各种各样的网络和多元环境中，它覆盖了所有片级别以上的语法级别，同时支持以下功能：支持独立片解码；起始码唯一保证；支持 SEI；支持流格式编码数据传送。

总体来说，NAL 解码器负责将符合 H.264 码流规范的压缩视频流解码，并进行图像重建。根据图 8.1 中的解码器框图，我们可以看到基本的解码流程如下：解码器从 NAL 中接收压缩的比特流，经过对码流进行熵解码获得一系列量化系数  $X$ ；这些系数经过反量化和反变换得到残差数据  $D$ ；解码器使用从码流中解码得到的头信息创建一个预测块  $PRED$ ， $PRED$  与残差数据  $D$  求和得到图像块数据  $uF$ ；最后每个  $uF$  通过去方块滤波得到重建图像的解码块  $F$ 。

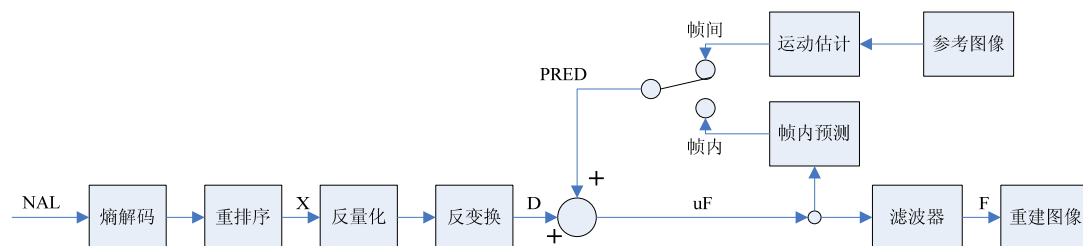


图 8.1 解码器功能框图

为更清晰的描述解码器的工作流程，我们用如图 8.2 的流程图来描述一帧图像的完整解码过程。

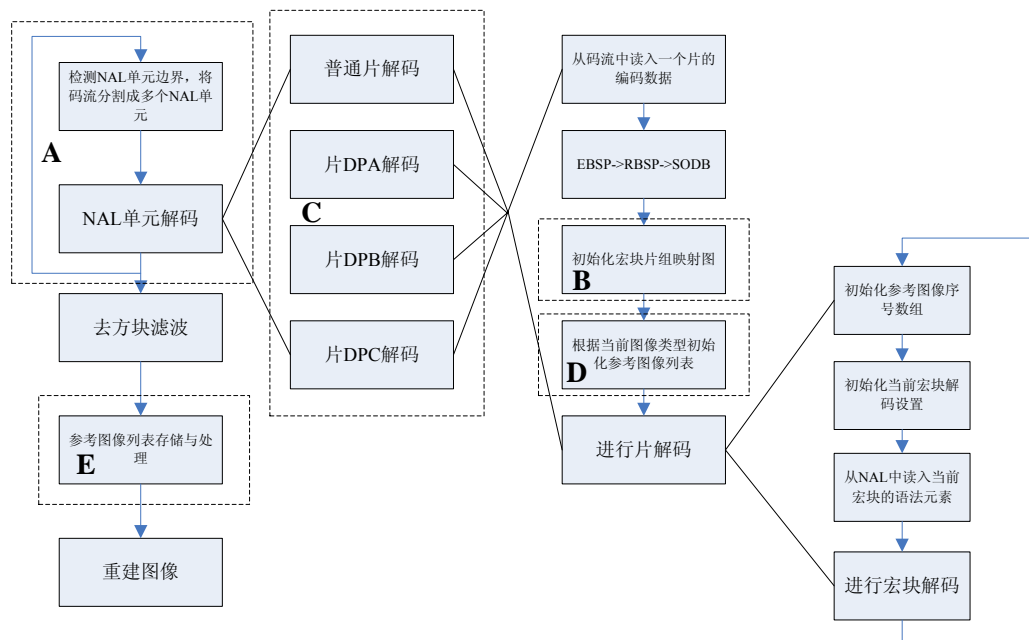


图 8.2 解码流程图

在图 8.2 中，A 框中的 NAL 单元边界检测和单元解码在 8.2 节中描述；8.3 节中将详细介绍解码过程中所使用到的一个重要参数，参考图像序列号（POC）的计算方法；8.4 节将阐述 B 框中宏块片

组映射图的产生过程；C 框中的片数据分割的解码过程描述在 8.5 节；D 框中的参考图像列表初始化和 E 框中参考图像存储和列表后处理等过程将在 8.6 节和 8.7 节中详细介绍。

## 8.2 NAL 单元

早期的视频压缩标准总是集中于比特流的概念，高层语法元素被编码分割，以允许比特流发生误码时进行重同步。而在 H.264/AVC 标准中，NAL (Network Abstract Layer) 则是以 NALU(NAL unit) 为单元来支持编码数据在基于包交换技术网络中的传输的；它定义了符合传输层或存储介质需求的数据格式，同时提供头信息，从而提供了视频编码与外部世界的接口。网络层和传输层的 RTP 封装只针对基于 NAL 单元的本地 NAL 接口，且每个 NAL 单元都只包含整数个字节。

### 8.2.1 NAL 单元结构

一个 NAL 单元定义了可用于基于包和基于比特流系统的基本格式，区别这两种格式的方法在于每个比特流传输层都有一个起始代码。在 NAL 解码器接口，它假定按传输顺序传递 NALU，同时，在 NALU 的头部设置标识接收正确的、丢失的或错误的标识位，如果在有效载荷中包含位错误，则通过标识位来标志。

一个 NAL 单元结构上是一个包含一定语法元素的可变长字节字符串，例如 NAL 单元可以携带一个编码片，A、B、C 型数据分割或一个序列或图像参数集。每个 NAL 单元由一个字节的头和一个包含可变长编码符号的字符串组成。头部含三个定长比特区，如图 8.3 所示：NALU 类型 (T)，NAL-REFERENCE-IDC (R) 和隐藏比特位 (F)。NALU 类型用 5bit 来代表 NALU 的 32 种中不同类型特征，类型 1—12 是 H.264 的定义的，类型 24—31 是用于 H.264 以外的，RTP 负荷规范使用这其中的一些值来定义包聚合和分裂，其他值为 H.264 保留。R 比特用于在重构过程中标记一个 NAL 单元的重要性，值为 0 表示这个 NAL 单元没有用于预测，因此可被解码器抛弃而不会有错误扩散，值高于 0 表示此 NAL 单元要用于无漂移重构，且值越高，对此 NAL 单元丢失的影响越大。最后是隐藏比特位，在 H.264 编码中默认置为 0，当网络识别到单元中存在比特错误时，可将其置为 1。F 位主要用于适应不同种类的网络环境(比如有线无线相结合的环境)。例如对于从无线到有线的网关，一边是无线的非 IP 协议环境，一边是有线网络的无比特错误的环境。假设一个 NAL 单元到达无线那边时，校验和检测失败，网关可以选择从 NAL 流中去掉这个 NALU 单元，也可以把已知被破坏的 NAL 单元前传给接收端。在这种情况下，智能的解码器将尝试重构这个 NAL 单元（已知它可能包含比特错误）。而非智能的解码器将简单的抛弃这个 NAL 单元。

1	2	3	4	5	6	7	8
T					R		F

图 8.3 NALU 头结构

### 8.2.2 NAL 单元解码过程

在进行 NAL 单元解码之前，首先或者通过 RTP 协议解析（采用 RTP 封装），或者通过起始码检测（采用比特流方式），从传输码流中获取 NAL 单元数据。NAL 单元解码的总体流程是：首先从 NAL 单元中提取出 RBSP 语法结构，然后按照如图 8.4 所示的流程处理 RBSP 语法结构。因此对于 NAL 单元的解码过程，其输入是 NAL 单元，输出结果是经过解码的当前图像（CurrPic）的样点值。



在 H.264 规范文档中规定：对于同一码流，所有的解码器必须产生数值上相同的结果，且必须符合规范定义的解码过程的标准。

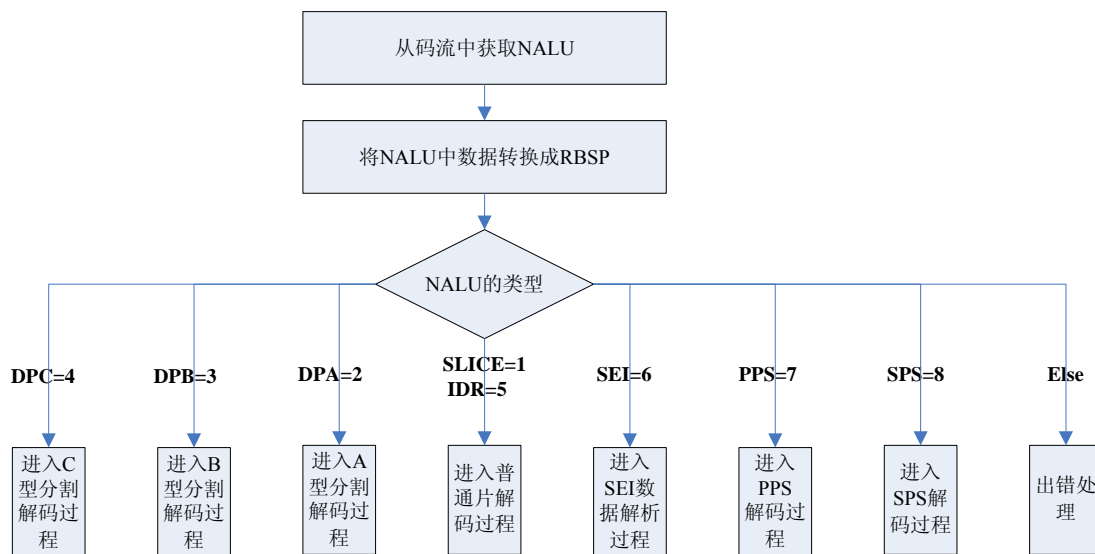


图 8.4 NAL 单元解码

在图 8.4 中，类型为 7、8 的 NAL 单元中分别包含了序列参数集和图像参数集。图像参数集和序列参数集在其它数据 NAL 单元的解码过程中作为参数使用，在这些数据 NAL 单元的片头中通过语法元素 `pic_parameter_set_id` 设置它们使用的图像参数集编号，而相应的每个图像参数集中通过语法元素 `seq_parameter_set_id` 设置它所使用的序列参数集编号。

## 8.3 图像序列号（picture order count）的计算

### 8.3.1 图像序列号（POC）

在 H.264 中，**图像序列号（POC）**主要用于标识图像的播放顺序，同时还用于在对帧间预测片解码时，标记参考图像的初始图像序号，表明下列情况下帧或场之间的图像序号差别：使用时间直接预测模式的运动矢量推算时；B 片中使用固有模式加权预测时；解码器一致性检测时。

其中**对于每个编码帧有两个图像序列号，分别称为顶场序列号（TopFieldOrderCnt）和底场序列号（BottomFieldOrderCnt）**；对于每个编码场有一个图像序列号，对于一个编码顶场其称为 TopFieldOrderCnt，对于编码底场，其称为 BottomFieldOrderCnt；对于每个编码场对有两个图像序列号，TopFieldOrderCnt 和 BottomFieldOrderCnt 分别用于标记该场对的顶场和底场。TopFieldOrderCnt 和 BottomFieldOrderCnt 分别指明了相应的顶场/底场相对于前一个 IDR 图像（或解码顺序中前一个包含 `memory_management_control_operation=5` 的参考图像）的第一个输出场的相对位置。

由于 H.264 使用了 B 帧预测，使得图像的解码顺序并不一定等于播放顺序，但它们之间存在一定的映射关系。H.264 中一共定义了三种 POC 的编码方法，句法元素 `pic_order_cnt_type` 就是用来通知解码器该用哪种方法来计算 POC，下面我们称之为 POC 类型。TopFieldOrderCnt 和 BottomFieldOrderCnt 根据当前序列参数集中的语法元素 `pic_order_cnt_type`（POC 类型）的值不同，按照如图 8.5 的计算流程获得。

如当前图像的语法元素 `memory_management_control_operation` 等于 5 时，在当前图像解码完成后，进行图像序列号的重新初始化操作，即将顶场和底场的 POC 均减去

PicOrderCnt( CurrPic)。 CurrPic 标志当前图像。函数 PicOrderCnt( )的实现参见图 8.6。

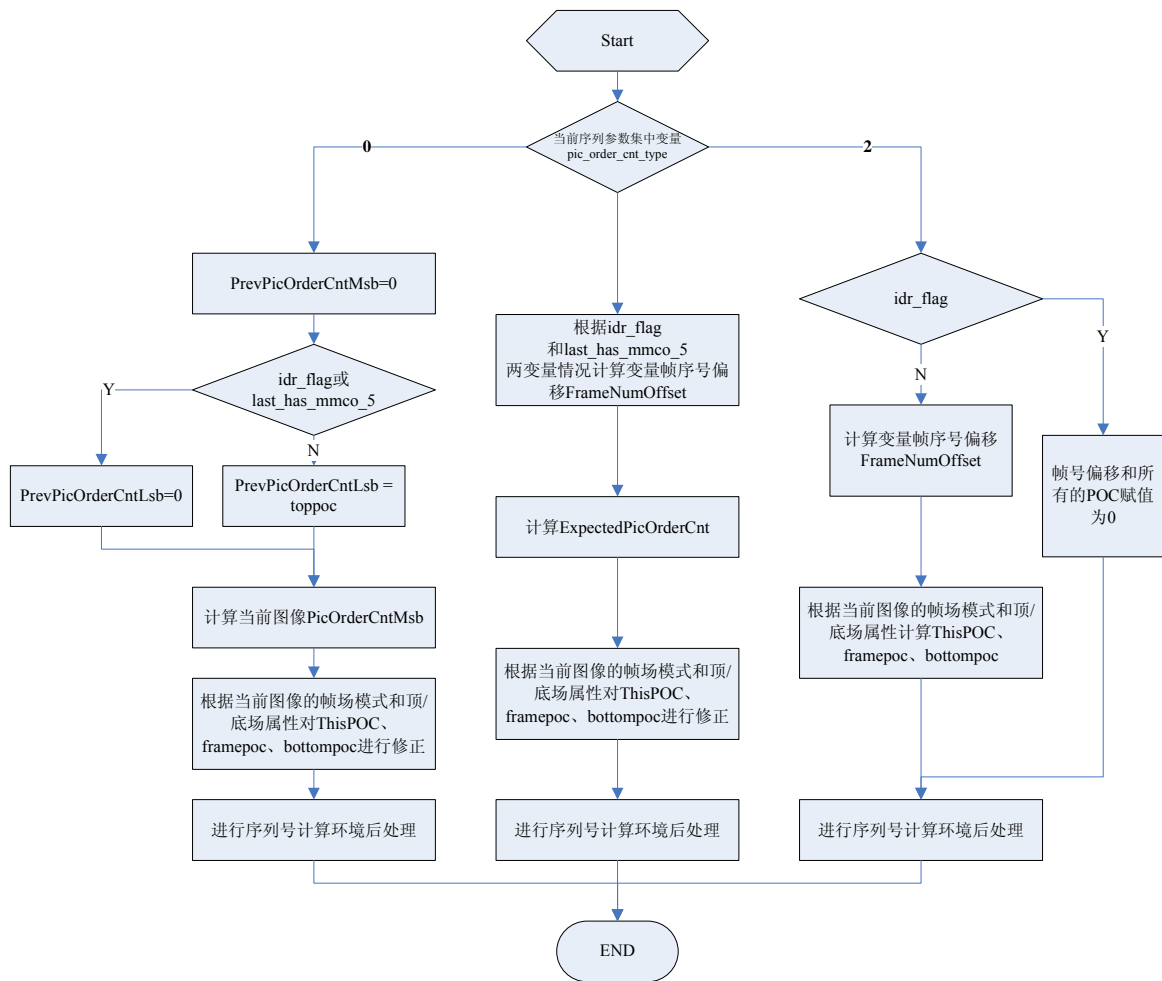


图 8.5 图像序列号的计算

对于图像序列号 POC，H.264 规范规定在 H.264 码流中不会出现下列情况：

- 一个IDR帧的Min( TopFieldOrderCnt, BottomFieldOrderCnt )不等于0；
- 一个IDR顶场的TopFieldOrderCnt不等于0；
- 一个IDR底场的BottomFieldOrderCnt不等于0。

因此对于一个编码 IDR 帧的两个场，TopFieldOrderCnt 和 BottomFieldOrderCnt 中至少有一个等于 0。

同时规范规定码流中不允许存在数据导致 TopFieldOrderCnt， BottomFieldOrderCnt， PicOrderCntMsb， 或 FrameNumOffset 超出  $-2^{31}$  到  $2^{31}-1$  的范围。如我们使用 DiffPicOrderCnt( picA, picB )表明两幅图像播放的时间间隔，即  $\text{DiffPicOrderCnt}( \text{picA}, \text{picB} ) = \text{PicOrderCnt}( \text{picA} ) - \text{PicOrderCnt}( \text{picB} )$ ，这样码流中的所有数据必须满足 DiffPicOrderCnt( picA, picB )的取值范围  $-2^{15}$  到  $2^{15}-1$ 。理由如下：假设 X 是当前图像， Y 和 Z 是同一序列中的另两个图像， Y 和 Z 被认为是从 X 起始的相同输出顺序方向，当 DiffPicOrderCnt( X, Y ) 和 DiffPicOrderCnt( X, Z ) 均为正或负时，其和不能超出  $-2^{31}$  到  $2^{31}-1$  的范围。

根据当前图像性质的不同，其使用的图像序列号 POC 会有所区别，因此使用函数 PicOrderCnt( picX )标记图像 picX 的图像序列号，很多应用中 PicOrderCnt( X )正比于图像 X 的采样时间与 IDR 采样时间的差值，其函数如图 8.6 中定义：

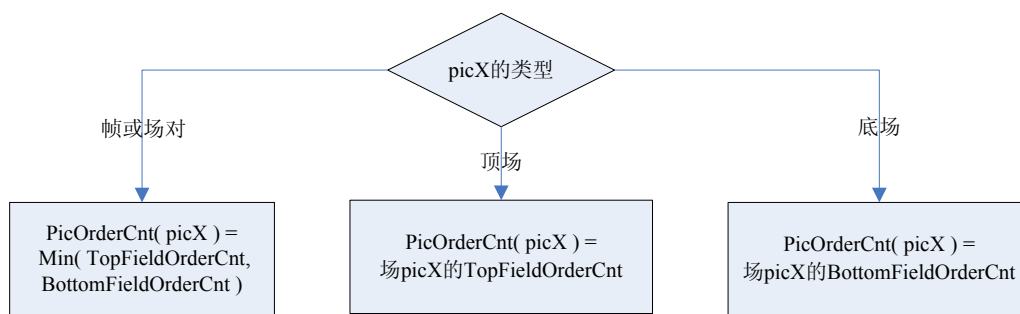


图 8.6 PicOrderCnt( picX )函数实现

下面我们分别介绍 POC 类型分别为 0, 1 和 2 的情况下, POC 的计算过程。

### 8.3.2 POC 类型为 0 的 POC 计算

当 `pic_order_cnt_type` 等于 0 时, 基于前一个参考图像 (按照解码顺序) 的 `PicOrderCntMsb` 计算当前图像的 `TopFieldOrderCnt` 和 (或) `BottomFieldOrderCnt`。首先计算变量 `prevPicOrderCntMsb` (前一个参考图像的 `PicOrderCntMsb`), 然后计算当前图像的 `PicOrderCntMsb`, 最后计算当前图像的 `TopFieldOrderCnt` 和 (或) `BottomFieldOrderCnt`。具体的计算流程参见图 8.7。

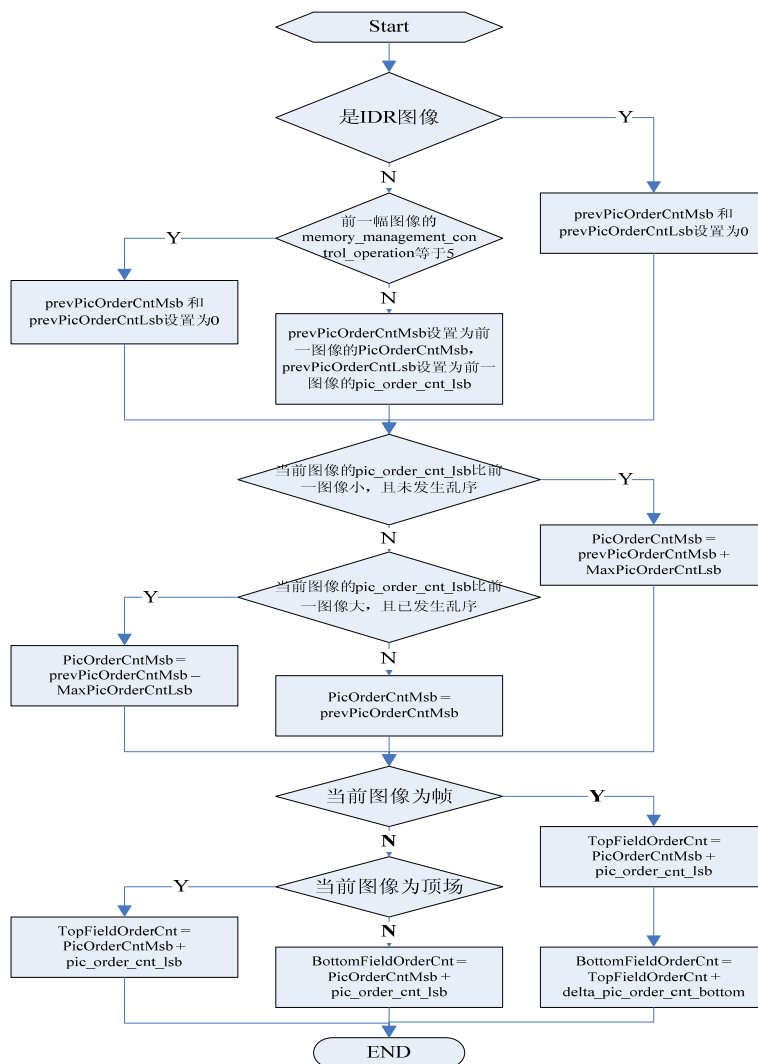


图 8.7 POC 类型=0 时 POC 的计算流程

### 8.3.3 POC 类型为 1 的 POC 计算

当 `pic_order_cnt_type` 等于 1 时，`TopFieldOrderCnt` 和（或）`BottomFieldOrderCnt` 按照流程图 8.8 计算。主要基于前一图像（按照解码顺序）的 `FrameNumOffset`，来计算 `TopFieldOrderCnt` 和（或）`BottomFieldOrderCnt`。计算过程中涉及到两个变量 `prevFrameNum` 和 `prevFrameNumOffset`，其中 `prevFrameNum` 是前一图像的 `frame_num`，而对于 `prevFrameNumOffset`，如当前图像不是 IDR，而前一图像的 **memory\_management\_control\_operation** 等于 5，`prevFrameNumOffset` 设为 0；否则，`prevFrameNumOffset` 设置等于前一图像的 `FrameNumOffset`。

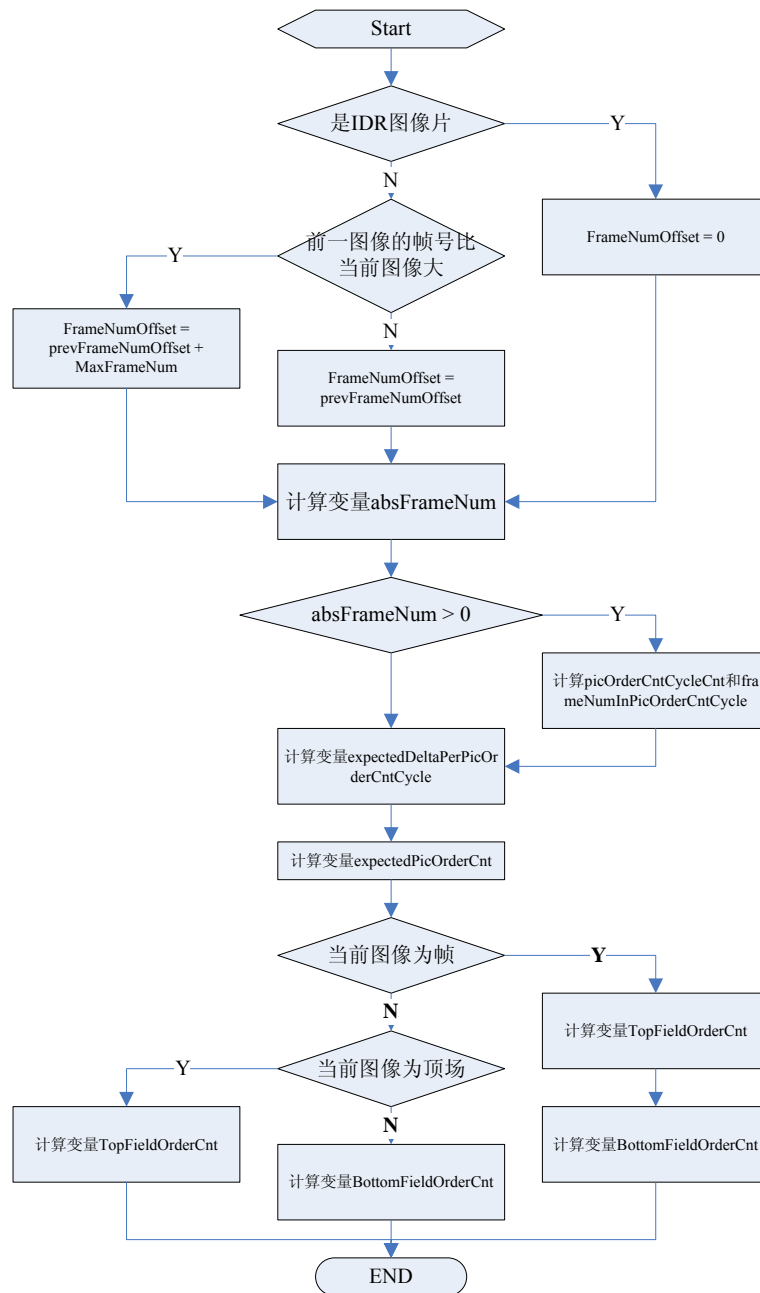


图 8.8 POC 类型=1 时 POC 的计算流程

### 8.3.4 POC 类型为 2 的 POC 计算

当 `pic_order_cnt_type` 等于 2 时，`TopFieldOrderCnt` 和（或）`BottomFieldOrderCnt` 的计算流程如图 8.9。Picture order count 类型 2 不能用于包含连续非参考图像的序列中，且解码结果导致输出的顺序与解码顺序相同。流程图中 `prevFrameNum` 表示按照解码顺序的前一图像的 `frame_num`。如当前图像不是 IDR 图像，而前一图像的 `memory_management_control_operation` 等于 5，`prevFrameNumOffset` 设为 0；否则，`prevFrameNumOffset` 设置等于前一图像的 `FrameNumOffset`。

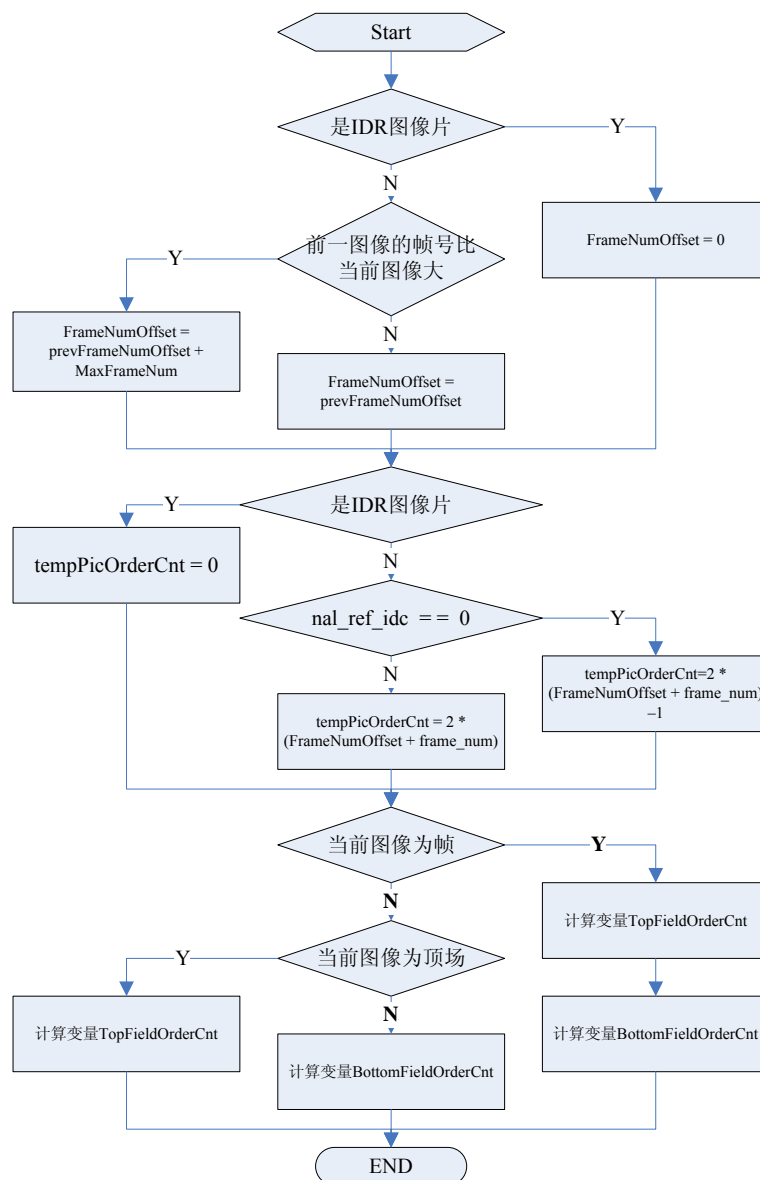


图 8.9 POC 类型=2 时 POC 的计算流程

## 8.4 宏块片组映射图的产生

解码器在对每个片解码之前，首先需要基于当前有效图像参数集和需解码的片头，产生宏块片组映射图变量 `MbToSliceGroupMap`，且此变量对于一个接入单元中的所有片均有效。当 `num_slice_groups_minus1` 等于 1 而且 `slice_group_map_type` 等于 3, 4, 或 5 时，片组 0 和 1 的大小

和形状根据表 8.1 由 slice\_group\_change\_direction\_flag 确定。

表 8.1 片组图类型

片组图类型	片组变化方向标记	说明
3	0	顺时针盒状展开
3	1	逆时针盒状展开
4	0	光栅扫描
4	1	逆光栅扫描
5	0	向右擦除
5	1	向左擦除

此时，按照定义的增长顺序的片组图单元 MapUnitsInSliceGroup0 被分配给片组 0，剩余的片组图单元 PicSizeInMapUnits – MapUnitsInSliceGroup0 被分配给片组 1。

当 num\_slice\_groups\_minus1 等于 1 且 slice\_group\_map\_type 等于 4 或 5 时，标记左上片组大小的变量 sizeOfUpperLeftGroup 这样定义：当片组变化方向标记=1 时，等于 PicSizeInMapUnits – MapUnitsInSliceGroup0，否则等于 MapUnitsInSliceGroup0。

变量 mapUnitToSliceGroupMap 的计算方法参见流程图 8.10。

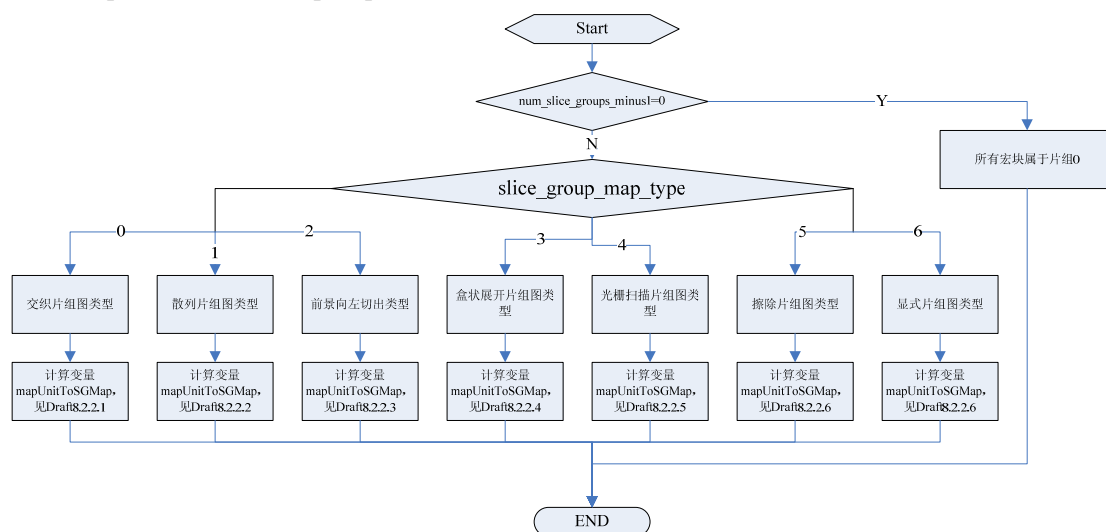


图 8.10 变量 mapUnitToSliceGroupMap 的计算流程

在图 8.10 的前景左切出类型中，在从 0 到 PicSizeInMapUnits – 1 之间至少存在一个 i，满足 mapUnitToSliceGroupMap[i] 等于 iGroup（对于每个从 0 到 num\_slice\_groups\_minus1 的 iGroup 值）即每个片组中至少包含一个片组图单元。此时片组覆盖区域的矩形可能重叠。片组 0 包含处于 top\_left[0] 和 bottom\_right[0] 定义的矩形内部的所有宏块。一个 ID 大于 0 且小于 num\_slice\_groups\_minus1 的片组包含所定义的矩形所覆盖的宏块，且这些宏块没有被 ID 较小的片组覆盖。ID 等于 num\_slice\_groups\_minus1 的片组包含没有被别的片组覆盖的宏块。

计算出 mapUnitToSliceGroupMap 后，会调用下面将要描述的片组图单元转换成宏块片组图的过程将片组图单元 mapUnitToSliceGroupMap 转换成宏块片组映射图 MbToSliceGroupMap，完成转换后，调用函数 NextMbAddress(n) 计算变量 nextMbAddress，函数流程如图 8.11。

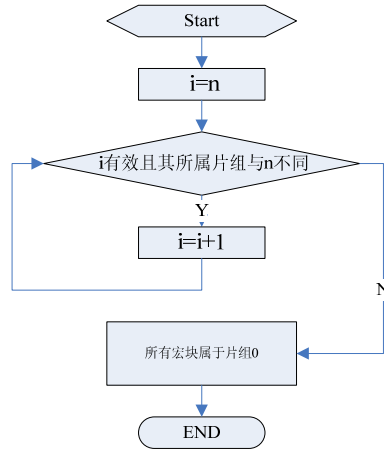


图 8.11 函数 NextMbAddress( n ) 的流程

在片组图单元转换成宏块片组图的过程规定如下：

- 如frame\_mbs\_only\_flag等于1或field\_pic\_flag等于1，宏块片组图设置为：

$$\text{MbToSliceGroupMap}[i] = \text{mapUnitToSliceGroupMap}[i] \quad (8.1)$$

- 如MbaffFrameFlag等于1，宏块片组图设置为：

$$\text{MbToSliceGroupMap}[i] = \text{mapUnitToSliceGroupMap}[i / 2] \quad (8.2)$$

- 否则（即frame\_mbs\_only\_flag等于0且mb\_adaptive\_frame\_field\_flag等于0且field\_pic\_flag等于0），宏块片组图设置为：

$$\begin{aligned} \text{MbToSliceGroupMap}[i] = & \text{mapUnitToSliceGroupMap}[(i / (2 * \text{PicWidthInMbs})) * \\ & \text{PicWidthInMbs} + (i \% \text{PicWidthInMbs})] \end{aligned} \quad (8.3)$$

其中 i 表示从 0 到 PicSizeInMbs – 1 的每个值。

## 8.5 片数据分割的解码

解码器根据一个片的 A 型数据分割 RBSP，如 NAL 类型为 3 或 4 时，还有一个属于同一片的 B 型数据分割 RBSP 或 C 型数据分割 RBSP，来产生一个完整的编码片。

当不使用片数据分割时，编码片仅有一个片层，没有数据分割 RBSP。分割 RBSP 包含一个片头，后跟包含了片中宏块数据的分类 2，3，4 语法元素的一个片数据语法结构。

当使用片数据分割时，片中的宏块数据分割成 1 到 3 个分割并位于单独的 NAL 单元中。分割 A 包括一个片数据分割 A 头部和所有的第 2 类语法元素。分割 B 包括一个片数据分割 B 头部和所有的第 3 类语法元素。分割 C 包括一个片数据分割 C 头部和所有的第 4 类语法元素。其中第 3 类语法元素与 I 和 SI 宏块类型的残差数据的解码有关。第 4 类语法元素与 P 和 B 宏块类型的残差数据的解码有关。第 2 类包含了所有其它的宏块解码相关的语法元素，这些信息通常称为头信息。片数据分割 A 的头部包含了片头的所有语法元素，还有一个 slice\_id 用于关联相应的分割 B 和分割 C。分割 B 和 C 的头只包含 slice\_id，用于建立与相应得分割 A 的联系。

当使用数据分割时，每类语法元素从单独的 NAL 单元中解析出来，如某类语法元素不存在，则相应的 NAL 单元不需出现。此时片数据分割的解析过程与没有分割 RBSP 类似，只需根据片数据分割的类型读出相应的语法元素。

## 8.6 参考图像列表的初始化

解码器在对每个 P、SP 或 B 片解码之前需要进行参考图像列表的初始化，从而生成参考图像列表 RefPicList0，当对 B 片解码时，还将生成另一个参考图像列表 RefPicList1。在参考图像列表中，解过码的参考图像根据码流的规定被标记成"used for short-term reference"或"used for long-term reference"，其中短期参考图像使用 frame\_num 的值作为标志，而长期参考图像则被分配一个长期参考帧索引，根据码流中的语法元素设定。

下面将对如下几点进行描述：

- 对每个短期参考帧，变量FrameNum和FrameNumWrap如何赋值；
- 每个短期参考图像的变量PicNum 如何计算；
- 长期参考图像的变量LongTermPicNum如何计算。

参考图像通过参考索引来进行标志。一个参考索引是一个变量 PicNum 和 LongTermPicNum 的数组下标，此数组称为参考图像列表。

如果 RefPicListX[i]使用 LongTermPicNum（对于一个长期参考图像），将 LongTermEntry（RefPicListX[i]）设置为 1；如果 RefPicListX[i]使用 PicNum（对于一个短期参考图像），则设为 0。其中 RefPicListX[i]标记参考图像列表 X 的第 i 个分量，其中 X=0 或 1。

在每个片开始解码时，参考图像列表 RefPicList0，对于 B 片有 RefPicList1 如下处理：

- 初始参考列表RefPicList0和B片的RefPicList1根据8.6.2中描述计算；
- 初始参考列表RefPicList0和B片的RefPicList1根据8.6.3中描述调整。

参考图像列表RefPicList0的下标分量有 num\_ref\_idx\_l0\_active\_minus1 + 1 个，B 片的 RefPicList1 下标分量有 num\_ref\_idx\_l1\_active\_minus1 + 1 个。一个参考图像可在 RefPicList0 或 RefPicList1 中出现一次以上。

### 8.6.1 图像序号的计算

变量 FrameNum，FrameNumWrap，PicNum，LongTermFrameIdx 和 LongTermPicNum 用于参考图像列表的初始化过程（见 2），调整过程（见 3）和解码参考图像标记过程（见？？）。

对于每个短期参考图像，变量 FrameNum 和 FrameNumWrap 的计算过程如下。首先，FrameNum 设置成相应短期参考图像的片头中解码所得语法元素 frame\_num，然后变量 FrameNumWrap 这样计算：如 FrameNum 大于当前图像片头中的 frame\_num，则 FrameNumWrap 等于 FrameNum 减去 MaxFrameNum；否则 FrameNumWrap 等于 FrameNum。

对于每个长期参考图像，变量 LongTermFrameIdx 按照 8.7 中的方法计算。

对于每个短期参考图像，分配一个 PicNum 变量，对于每个长期参考图像，分配一个 LongTermPicNum 变量。这些变量的值由当前图像的 field\_pic\_flag 和 bottom\_field\_flag 的值决定，它们的设置见图 8.12。



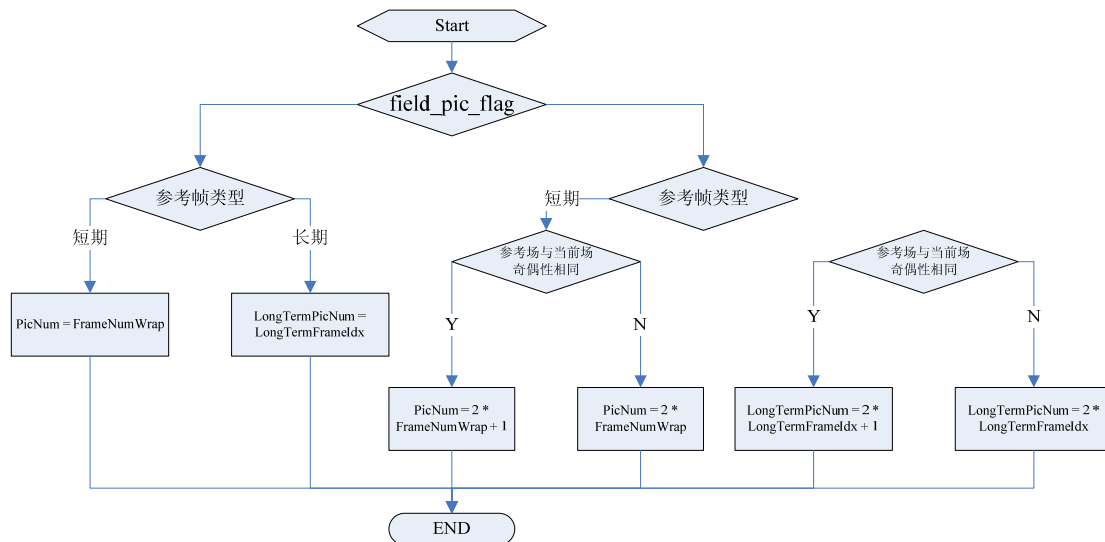


图 8.12 PicNum 变量的计算

## 8.6.2 参考图像列表的初始化

解码器在对 P，SP，或 B 片片头解码时进行参考图像列表的初始化，生成初始化好的参考图像列表 RefPicList0，和 B 片解码时所需的 RefPicList1。RefPicList0 和 RefPicList1 的各分量使用变量 PicNum 和 LongTermPicNum 初始化，如 8.6.2.1-8.6.2.5 所述。

当 8.6.2.1-8.6.2.5 中产生的 RefPicList0 或 RefPicList1 中的分量个数分别大于 num\_ref\_idx\_l0\_active\_minus1 + 1 或 num\_ref\_idx\_l1\_active\_minus1 + 1 时，多余的分量将从列表中丢弃；如果情况相反，分量个数分别小于 num\_ref\_idx\_l0\_active\_minus1 + 1 或 num\_ref\_idx\_l1\_active\_minus1 + 1 时，列表中剩余的分量设置为"no reference picture"。

### 8.6.2.1 帧中 P 和 SP 片的参考帧列表的初始化

解码器在对一个编码帧中的 P 或 SP 片解码之前通过参考帧列表的初始化生成 RefPicList0。RefPicList0 是排序的，短期参考帧和短期补充参考场对位于长期参考帧和长期参考场对之前；同时，短期参考帧和短期补充参考场对根据 PicNum 值进行降序排列；长期参考帧和长期参考场对根据 LongTermPicNum 值进行升序排列。同时在 H.264 中对帧进行解码时，不论 MbaffFrameFlag 取什么值，帧间预测不使用不成对的参考场。

例如，当三个参考帧使用 PicNum=300，302，303 标记为"用于短期参考"，两个参考帧用 LongTermPicNum=0，3 标记为"用于长期参考"：

- RefPicList0[0] 设置为 PicNum = 303,
- RefPicList0[1] 设置为 PicNum = 302,
- RefPicList0[2] 设置为 PicNum = 300,
- RefPicList0[3] 设置为 LongTermPicNum = 0,
- RefPicList0[4] 设置为 LongTermPicNum = 3.

且 LongTermEntry(RefPicList0[i]) 设置为 0，当 i=0, 1, and 2；设置为 1，当 i=3，4。

### 8.6.2.2 场中 P 和 SP 片的参考帧列表的初始化

解码器在对一个编码场中的 P 或 SP 片解码之前同样需要首先进行 RefPicList0 的初始化，不同的是对一个场解码时，参考图像列表中的每个场都有单独的列表索引；而且对一个场解码时，可用的参考图像数将是解码一个帧时的两倍。在此过程中，首先需要计算两个有序的参考帧列表 refFrameList0ShortTerm 和 refFrameList0LongTerm；为方便参考帧列表排列，解码的帧，参考场对，不成对参考场，和一个场标记为“用于短期参考”或“用于长期参考”的参考帧均被认为是参考帧。这两个列表的产生过程如下：

- 所有的有一个或多个场标记为“用于短期参考”的帧的 FrameNumWrap 包含在短期参考帧列表 refFrameList0ShortTerm 中。如当前场是参考场对的第二个场（按照解码顺序）且第一个场标记为“用于短期参考”，当前场的 FrameNumWrap 也包含在 refFrameList0ShortTerm 中。refFrameList0ShortTerm 按照 FrameNumWrap 值的降序进行排列；
- 所有的有一个或多个场标记为“用于长期参考”的帧的 LongTermFrameIdx 包含在长期参考帧列表 refFrameList0LongTerm 中。如当前场是参考场对的第二个场（按照解码顺序）且第一个场标记为“用于长期参考”，第一个场的 LongTermFrameIdx 也包含在 refFrameList0LongTerm。refFrameList0LongTerm 按照 LongTermFrameIdx 值的升序进行排列。

在这两个列表产生后，将根据在 8.6.2.5 中定义的过程，使用 refFrameList0ShortTerm 和 refFrameList0LongTerm 作为输入，生成 RefPicList0。

### 8.6.2.3 帧模式的 B 片参考图像列表的初始化

与 8.6.2.1、8.6.2.2 不同，解码器在对一个编码帧的 B 片解码时首先需要对参考帧列表 RefPicList0 和 RefPicList1 进行初始化。对于 B 片，RefPicList0 和 RefPicList1 中短期参考图像的排列顺序取决于输出次序，即由 PicOrderCnt( ) 给定。参考帧列表 RefPicList0 和 RefPicList1 都是有序排列的，其中短期参考帧和短期参考场对均排在长期参考帧和长期参考场对之前。

参考帧列表 RefPicList0 详细的排列顺序如下：

- 短期参考帧和短期参考场对排列顺序如下：首先是 PicOrderCnt( CurrPic )-PicOrderCnt( frm0 ) 值最大的短期参考帧或参考场对 frm0，按照降序排列到 PicOrderCnt( frm1 ) 最小的短期参考帧或参考场对 frm1，然后是 PicOrderCnt( frm2 )-PicOrderCnt( CurrPic ) 值最小的短期参考帧或参考场对 frm2，然后按照升序排列到 PicOrderCnt( frm3 ) 值最大的短期参考帧或参考场对；
- 长期参考帧和长期参考场对的顺序按照 LongTermPicNum 值的升序排列。

参考帧列表 RefPicList1 详细的排列顺序如下：

- 短期参考帧和短期参考场对排列顺序如下：首先是 PicOrderCnt( frm4 )-PicOrderCnt( CurrPic ) 值最小的短期参考帧或参考场对 frm4，按照升序排列到 PicOrderCnt( frm5 ) 最大的短期参考帧或参考场对 frm5，然后是 PicOrderCnt( CurrPic )-PicOrderCnt( frm6 ) 值最大的短期参考帧或参考场对 frm6，然后按照降序排列到 PicOrderCnt( frm7 ) 值最小的短期参考帧或参考场对 frm7；
- 长期参考帧和长期参考场对的顺序按照 LongTermPicNum 值的升序排列；
- 当参考图像列表 RefPicList1 有多于一个入口且它与 RefPicList0 相同时，头两个入口 RefPicList1[0] 和 RefPicList1[1] 将被交换。

#### 8.6.2.4 场模式下 B 片的参考帧列表初始化

同样解码器在对一个编码场的 B 片解码之前首先需要初始化参考帧列表 RefPicList0 和 RefPicList1。当对一个场解码时，存储的参考帧的每个场使用一个唯一的索引标记为一个单独的参考图像。RefPicList0 和 RefPicList1 中短期参考图像的顺序取决于输出次序，即由 PicOrderCnt() 给定。在此过程中，首先需要推导三个有序的参考帧列表 refFrameList0ShortTerm、refFrameList1ShortTerm 和 refFrameListLongTerm。在下面的描述中，用术语“参考条目”表示已解码参考帧、参考场对和非成对参考场。

refFrameList0ShortTerm 列表构建方法如下：首先是 PicOrderCnt( CurrPic )- PicOrderCnt( f0 )值最大的参考条目 f0，按照降序排列到 PicOrderCnt( f1 )最小的参考条目 f1，然后是 PicOrderCnt( f2 )- PicOrderCnt( CurrPic )值最小的短期参考帧或参考场对 f2，然后按照升序排列到 PicOrderCnt( f3 )值最大的参考条目 f3。须注意的是如当前场的 nal\_ref\_idc 大于 0 且根据解码顺序当前编码场前面是属于同一参考场对的编码场 fld1 时，fld1 必须用 PicOrderCnt( fld1 )包含在列表 refFrameList0ShortTerm 中，且需符合前面所述的排序规则。

refFrameList1ShortTerm 的构建构成如下：首先是 PicOrderCnt( f4 )- PicOrderCnt( CurrPic )值最小的参考条目 f4，按照升序排列到 PicOrderCnt( f5 )最大的短期参考条目 f5，然后是 PicOrderCnt( CurrPic )- PicOrderCnt( f6 )值最大的参考条目 f6，然后按照降序排列到 PicOrderCnt( f7 )值最小的参考条目 f7。如当前场的 nal\_ref\_idc 大于 0 且根据解码顺序当前编码场前面是属于同一参考场对的编码场 fld2 时，fld2 必须用 PicOrderCnt( fld2 )包含在列表 refFrameList1ShortTerm 中，且需符合前面所述的排序规则。

对于 refFrameListLongTerm，其列表次序按照 LongTermPicNum 值的升序排列。如当前图像的场对标记为“用于长期参考”，它将被包含在列表 refFrameListLongTerm 中。一个仅一个场标记为“用于长期参考”的参考条目也包含在列表 refFrameListLongTerm 中。

在三个列表生成后，根据 8.6.2.5 中描述的过程，使用 refFrameList0ShortTerm 和 refFrameListLongTerm 作为输入，生成 RefPicList0；使用 refFrameList1ShortTerm 和 refFrameListLongTerm 作为输入，生成 RefPicList1。当参考图像列表 RefPicList1 有多于一个入口且它与 RefPicList0 相同时，头两个入口 RefPicList1[0] 和 RefPicList1[1] 将被交换。

#### 8.6.2.5 场模式的参考帧列表的初始化

解码器在场模式下，对 P 片、SP 片和 B 片解码时，首先需要根据上面描述的过程生成参考帧列表 refFrameListXShortTerm (X 为 0 或 1) 和 refFrameListLongTerm；然后在以它们作为输入参数是，生成参考图像列表 RefPicListX (X 为 0 或 1)。

在参考图像列表 RefPicListX 中，短期参考场位于长期参考场之前。RefPicListX 的推导过程如下：

- 短期参考场的顺序根据已排序的帧列表 refFrameListXShortTerm 选择奇偶交替的场得到：首先是与当前场奇偶性相同的场。如一个参考帧的某个场未解码或未被标记为“用于短期参考”，丢失的场将被忽略，下一个从 refFrameListXShortTerm 选择的相同奇偶性的有效的存储参考场被插入 RefPicListX。如 refFrameListXShortTerm 中没有更多的奇偶交替的短期参考场，下一个未被索引的奇偶有效的场按照它们在 refFrameListXShortTerm 的顺序插入 RefPicListX；
- 长期参考场的顺序根据已排序的帧列表 refFrameListLongTerm 选择奇偶交替的场得到：首先是与当前场奇偶性相同的场。如一个参考帧的某个场未解码或未被标记为“用于长期参

考”，丢失的场将被忽略，下一个从 refFrameListLongTerm 选择的相同奇偶性的有效的存储参考场被插入 RefPicListX。如 refFrameListLongTerm 中没有更多的奇偶交替的长期参考场，下一个未被索引的奇偶有效的场按照它们在 refFrameListLongTerm 的顺序插入 RefPicListX。

### 8.6.3 参考帧列表的重排序

解码器在根据 8.6.2.2 中描述的过程产生参考图像列表 RefPicList0 和 RefPicList1 后，需要根据片头码流中的相关语法元素，如 ref\_pic\_list\_reordering\_flag\_l0、ref\_pic\_list\_reordering\_flag\_l1、reordering\_of\_pic\_nums\_idc、abs\_diff\_pic\_num\_minus1 和 long\_term\_pic\_num 等的规定，经过下面的重排序过程进行列表的重排序。重排序的流程如图 8.13 所示。

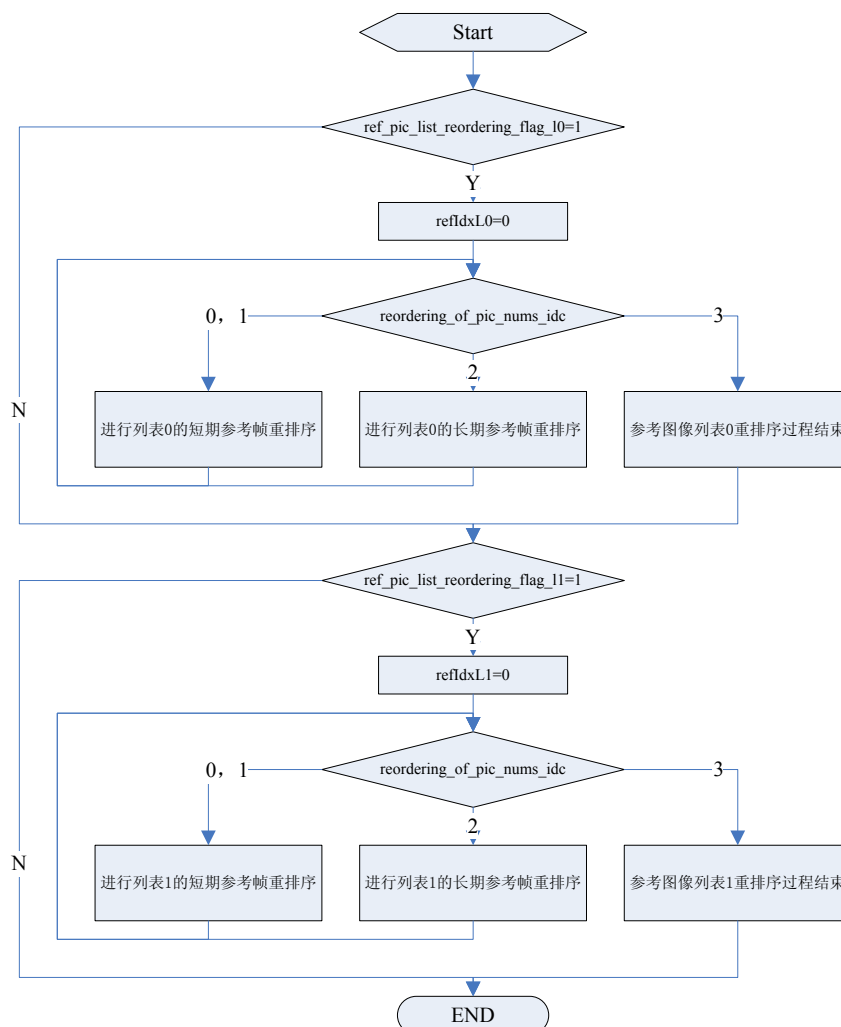


图 8.13 参考帧列表的重排序流程

其中 refIdxL0 是参考图像列表 RefPicList0 的一个索引，refIdxL1 是参考图像列表 RefPicList1 的一个索引，流程图中的短期参考帧重排序过程和长期参考帧重排序过程在 8.6.2.1 和 8.6.2.2 中详细描述。请注意，在短期参考帧的重排序和长期参考帧的重排序过程中，表 RefPicListX 的长度比最终列表需要的长度大 1；在排序过程执行完毕后，仅列表的元素 0 到 num\_ref\_idx\_lx\_active\_minus1 需要保留。

### 8.6.3.1 短期参考帧的重排序

短期参考帧的重排序过程的输入参数是参考图像列表 RefPicListX (X 为 0 或 1)和这个列表的一个索引 refIdxLX；输出是调整之后的参考图像列表 RefPicListX (X 为 0 或 1)和增加后的索引 refIdxLX。过程的实现流程参见图 8.14。

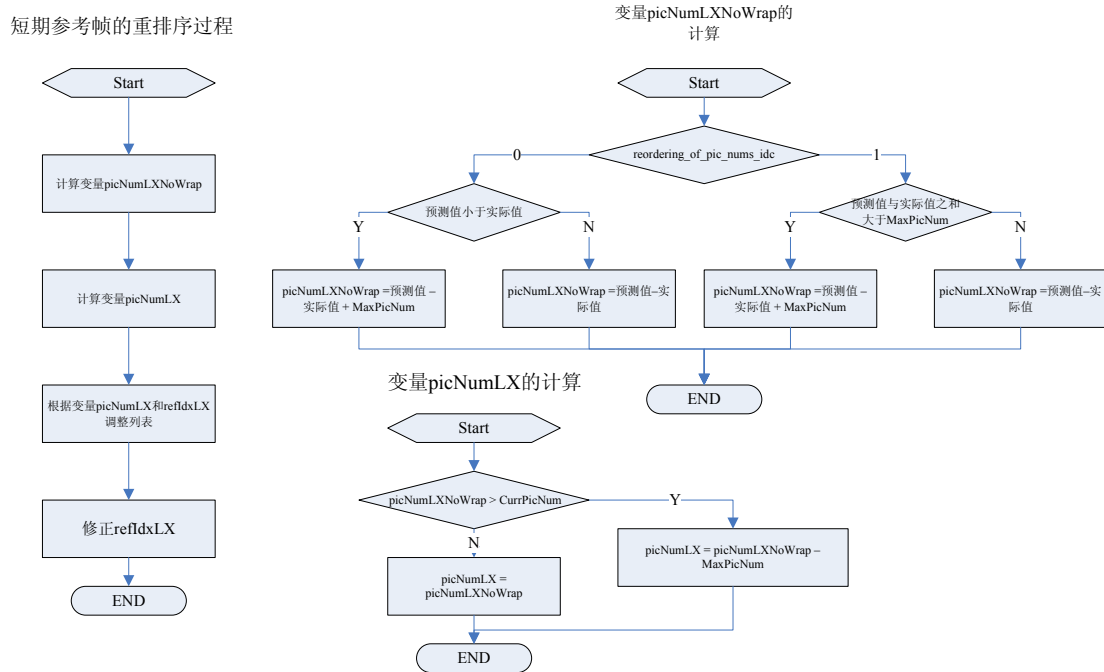


图 8.14 短期参考帧的重排序流程

其中图 8.14 中的预测值即 Draft 中的变量 picNumLXPred，是变量 picNumLXNoWrap 的预测值。当一个片第一次调用本过程时（即在 ref\_pic\_list\_reordering() 语法中第一次出现 reordering\_of\_pic\_nums\_idc 等于 0 或 1），picNumL0Pred 和 picNumL1Pred 初始设置等于 CurrPicNum。在每次 picNumLXNoWrap 的分配之后，picNumLXNoWrap 被赋给 picNumLXPred。

图中的 picNumLX 应指定一个标记为“用于短期参考”的参考图像，而不应指定一个标记为“不存在”的短期参考图像。需要执行下面的过程将短期图像序号为 picNumLX 的图像放入 refIdxLX 索引指定的位置，并将后面的其它图像向后移位，增加 refIdxLX 的值。

```

for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx-- )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx - 1]
RefPicListX[ refIdxLX++ ] = picNumLX
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )
    if( LongTermEntry( RefPicListX[ cIdx ] ) || RefPicListX[ cIdx ] != picNumLX )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
  
```

(8.4)

### 8.6.3.2 长期参考帧的重排序

长期参考帧的重排序过程的输入参数是参考图像列表 RefPicListX (X 为 0 或 1)和这个列表的一个索引 refIdxLX；输出是调整之后的参考图像列表 RefPicListX (X 为 0 或 1)和增加后的索引 refIdxLX。

重排序过程如下：首先根据图像的 LongTermPicNum 等于 long\_term\_pic\_num 这一条件，确定一个将标记为“用于长期参考”的参考图像；然后需要执行下面的过程将长期图像序号为 long\_term\_pic\_num 的图像放入 refIdxLX 索引指定的位置，并将后面的其它图像向后移位；最后修正 refIdxLX 的值。

```

for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx-- )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx - 1 ]
RefPicListX[ refIdxLX++ ] = LongTermPicNum

nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )
    if( !LongTermEntry( RefPicListX[ cIdx ] ) || RefPicListX[ cIdx ] != LongTermPicNum )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
(8.5)

```

## 8.7 解码的参考图像的标记过程

根据图 8.2 中的解码整体流程可以看到，解码器在完成一幅图像的解码后，需要对已解码图像进行存储处理，如果此图像用于其它图像的参考，同时还要进行已解码图像的标记操作。当 nal\_ref\_idc 等于 0 时，不需要进行此操作。也就是说一个 nal\_ref\_idc 不等于 0 的已解码图像，如是一个参考图像，将被标记为“用于短期参考”或“用于长期参考”。

标记为“用于短期参考”或“用于长期参考”的帧或参考场对可在帧解码时用于帧间预测的参考；标记为“用于短期参考”或“用于长期参考”的场可在场解码时用于帧间预测的参考；当一个帧或它的某个场标记为“不用于参考”，或一个场标记为“不用于参考”时，它们将从参考列表中删除。一个图像在参考图像滑窗标记过程（采用先进先出机制）或参考图像自适应内存控制标记过程（采用客户自适应标记机制）中，进行“不用于参考”的标记操作，这两个过程将在 8.7.1 和 8.7.2 描述。

解码器使用图像序号 PicNum 标记一个短期参考图像，使用长期图像序号 LongTermPicNum 标记一个长期参考图像。关于 PicNum 和 LongTermPicNum 计算方法在 8.6.1 中已描述。

已解码参考图像标记过程的操作流程参见图 8.15。

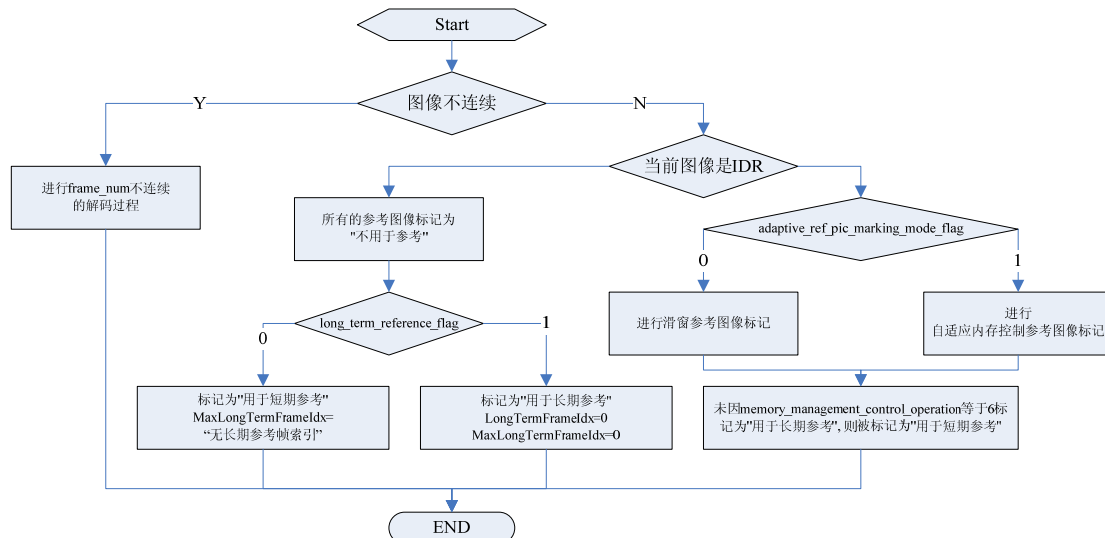


图 8.15 参考图像标记流程

图中的图像不连续表示，当前图像与前一幅图像之间出现跳跃，即 `frame_num` 不等于 `PrevRefFrameNum` 且不等于  $(\text{PrevRefFrameNum} + 1) \% \text{MaxFrameNum}$ 。此外在对当前图像标记完毕后，至少一个场标记为“用于参考”的帧数加上至少一个场标记为“用于参考”的场对数，加上标记为“用于参考”的非成对场数的和，不应大于 `num_ref_frames`。下面详细介绍 `frame_num` 不连续的解码过程、参考图像“滑窗”标记过程和参考图像自适应内存控制标记过程。

### 8.7.1 `frame_num` 不连续的解码过程

本过程只在 `gaps_in_frame_num_value_allowed_flag` 等于 1 的相应比特流中调用。如果比特流的 `gaps_in_frame_num_value_allowed_flag` 等于 0 且 `frame_num` 不等于 `PrevRefFrameNum` 或  $(\text{PrevRefFrameNum} + 1) \% \text{MaxFrameNum}$ ，解码过程应推断图像丢失已经发生。如本过程被调用，需要计算一系列“不存在”图像所附属的 `frame_num` 值，所有的值根据第 7 章中描述的变量 `UnusedShortTermFrameNum` 获得，除了当前图像的 `frame_num` 值。

解码过程需要对每个“不存在”图像所附属的 `frame_num` 值进行标记，按照 `UnusedShortTermFrameNum` 值的顺序，使用 8.7.2 中描述的“滑窗”图像标记过程。添加的帧必须也标记为“不存在”和“用于短期参考”。添加帧的采样值设置为任意值。这些标记为“不存在”的添加帧不能用于帧间预测过程，不能在短期参考图像参考图像列表的重排序命令中使用，不能在分配 `LongTermFrameIdx` 给一个短期参考图像的过程中使用。当一个标记为“不存在”的帧使用“滑窗”缓冲过程或“自适应内存控制”机制标记为“不用于参考”，它不应再被标记为“不存在”。

如果任何一个“不存在”图像所附属的 `frame_num` 值在帧间预测或参考索引分配中使用，解码过程应能推断图像丢失发生。如内存管理控制操作用到标记为“不存在”的帧，解码过程不必推断图像丢失发生。

### 8.7.2 参考图像滑窗标记过程

本过程当 `adaptive_ref_pic_marking_mode_flag` 等于 0 时调用。

如果当前编码场是一个参考场对中按照解码顺序的第二个场，且第一个场已被标记为“用于短期参考”，当前图像也标记为“用于短期参考”。否则：

- 将 `numShortTerm` 赋值为至少一个场标记为“用于短期参考”的参考帧、参考场对和非成对参考场的总数。将 `numLongTerm` 赋值为至少一个场标记为“用于长期参考”的参考帧、参考场对和非成对参考场的总数。
- 当 `numShortTerm + numLongTerm` 等于 `num_ref_frames` 时，必须满足 `numShortTerm` 大于 0 且有着最小 `FrameNumWrap` 值的短期参考帧、参考场对或非成对参考场必须标记为“不用于参考”。如它是一个帧或场对，那么它的两个场必须均标记为“不用于参考”。

### 8.7.3 参考图像的自适应内存控制标记过程

本过程当 `adaptive_ref_pic_marking_mode_flag` 等于 1 时调用。在当前图像解码完毕后，根据 `memory_management_control_operation` 命令，按照它们在比特流出现的顺序执行，其中命令参数为从 1 到 6。根据 `memory_management_control_operation` 的参数值，对于每个不同命令的具体操作，在(1)到(6)中详细描述。如参数值为 0 意味着操作结束。此外，操作过程中，解码器根据 `field_pic_flag` 的值确定操作的对象类型，如为 0，则针对指定的帧或场对；如为 1，则针对指定的某个场。具体操作过程参见流程图 8.16。

在将一个短期或长期参考场标记为“不用于参考”的情况下，另一个场的标记在本过程中不发生变化，但会在下一次调用本过程时处理，如同第七章中已提到。

在分配一个 LongTermFrameIdx 给一个短期参考图像过程中，根据给定语法元素 difference\_of\_pic\_nums\_minus1 的值计算变量 picNumX 的方法参照图 8.16。其中 picNumX 用来指定一个标记为“用于短期参考”且未被标记为“不存在”的帧、参考场对或非成对参考场。

在根据 MaxLongTermFrameIdx 值进行标记的过程中，所有的 LongTermFrameIdx 大于 max\_long\_term\_frame\_idx\_plus1 - 1 且标记为“用于长期参考”的图像标记为“不用于参考”。变量 MaxLongTermFrameIdx 的计算方法参见流程图 8.16。请注意发送 max\_long\_term\_frame\_idx\_plus1 的频度在当前建议/国际标准中没有定义；然而，编码器在接收到一个错误信息后必须发送一个 memory\_management\_control\_operation 命令 4，例如帧内刷新请求消息。

在将所有参考图像标记为“不用于参考”的过程中，必须首先将所有参考图像标记为“不用于参考”，然后将 MaxLongTermFrameIdx 设置为“无长期帧索引”。

在分配一个长期帧索引给当前图像的过程中，如果 LongTermFrameIdx 已经分配给一个长期参考帧或参考场对，相应的参考帧或场对及其两个场均先要标记为“不用于参考”。当这个 LongTermFrameIdx 已经被分配给一个非成对参考场，且这个场不是当前图像的互补场，这个场需被标记为“不用于参考”。然后将当前图像标记为“用于长期参考”且 LongTermFrameIdx 赋值为 long\_term\_frame\_idx。当 field\_pic\_flag 等于 0，它的两个场也标记为“用于长期参考”且 LongTermFrameIdx 赋值为 long\_term\_frame\_idx。当 field\_pic\_flag 等于 1 且当前图像是一个参考场对的第二个场（按照解码顺序），相应的场对也标记为“用于长期参考”，且 LongTermFrameIdx 赋值为 long\_term\_frame\_idx。



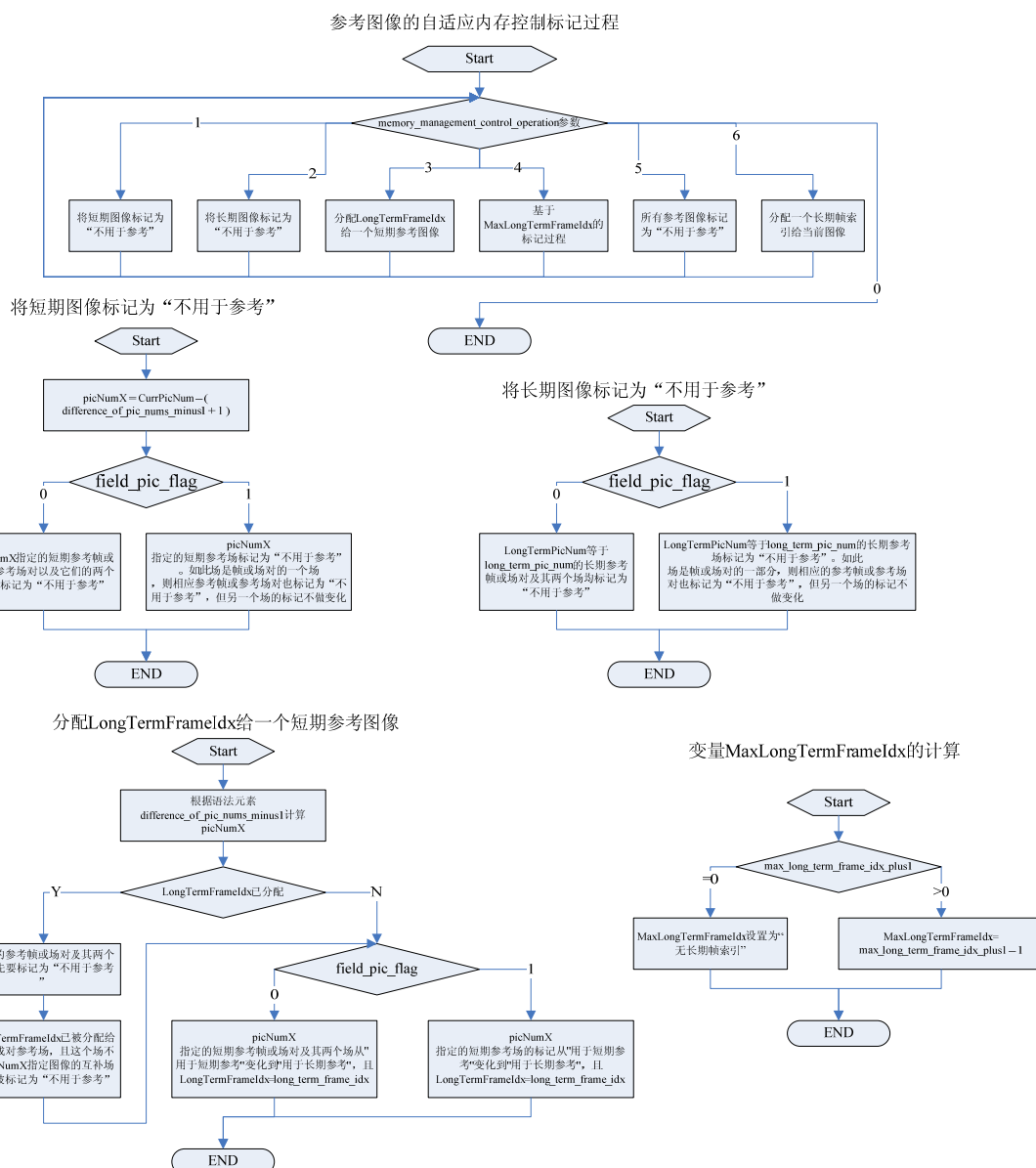


图 8.16 参考图像的自适应内存控制标记过程

## 8.8 帧内预测

视频压缩中所采用的帧内预测技术用于去除当前图像中的空间冗余度，新的编码标准 H.264/AVC 采用了比以往编码标准更为精确和复杂的帧内预测方式。在当前图像编码过程中，当无法提供足够的图像之间的时间相关信息时，往往对当前图像采用了帧内预测。由于当前被编码的宏块与相邻的宏块有很强的相似性，因此在 H.264/AVC 中的帧内预测用于计算编码宏块与其相邻宏块之间的空间相关性，以提高编码效率。在帧内预测中当前编码的宏块上方及左方的宏块用于计算出当前宏块的预测值。当前宏块与其预测值的差值将进一步编码并予传输到解码器端。解码器利用该比特流中用于表示预测方式相关比特与解码出的残差信号的比特，并计算出当前宏块的预测值，恢复图像宏块的原始像素值。H.264/AVC 也提供了“PCM”的编码方式，采用“PCM”编码时，解码器可直接恢复当前宏块的像素值而不必经过其他任何计算。

H.264/AVC 提供了四种帧内预测方式：4x4 亮度块的帧内预测（Intra\_4x4）、16x16 亮度块的帧

内预测(Intra\_16x16)、8x8 色度块的帧内预测(Intra\_chroma)，以及 PCM 的预测方式(I\_PCM)。

### 8.8.1 4x4 亮度块预测方式的提取

当前宏块采用 4x4 亮度块的帧内预测方式 (Intra\_4x4) 时每一个 4x4 块可使用如表 8.2 所示的 9 种预测方式的任意一种进行预测，各种预测方式的预测方向如图 8.15 所示。

表 8.2 4x4 亮度块的帧内预测模式

预测模式序号	预测模式
0	Intra_4x4_Vertical
1	Intra_4x4_Horizontal
2	Intra_4x4_DC
3	Intra_4x4_Diagonal_Down_Left
4	Intra_4x4_Diagonal_Down_Right
5	Intra_4x4_Vertical_Right
6	Intra_4x4_Horizontal_Down
7	Intra_4x4_Vertical_Left
8	Intra_4x4_Horizontal_Up

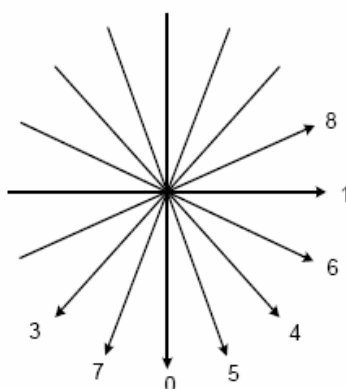


图 8.15 4x4 亮度块预测方式下的预测方向

然而全部传输每一个 4x4 亮度块的预测方式必将占用大量的比特数，因此需要考虑利用相邻块的预测方式之间的相关性。例如，已编码的 4x4 亮度块 A 及 B 位于当前块的上方及左方，若 A 和 B 均采用模式 2 的帧内预测方式，对于当前块 C 最有可能的预测方式也是模式 2。

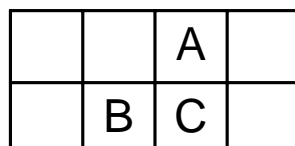


图 8.16 4x4 亮度块 C 的相对位置

对于当前的 4x4 亮度块 C，若 C 的预测方式与 A 与 B 的预测方式的最小值 (predIntra4x4Pred 模式) 相同，此时相应的句法单元 prev\_intra4x4\_pred\_模式置为 1。否则 (prev\_intra4x4\_pred\_模式置为 0)，另一个句法单元 rem\_intra4x4\_pred\_模式用于指出 C 块的预测方式与 predIntra4x4Pred 模式的差别。若 rem\_intra4x4\_pred\_模式 的值小于 predIntra4x4Pred 模式则当前块 C 的预测方式为

rem\_intra4x4\_pred\_模式 的值；否则当前块 C 的预测方式为 rem\_intra4x4\_pred\_模式+1 的值。此外若图 8.16 中的 A 或 B 不存在，则其相应的预测方式为直流预测（模式 2）。

### 8.8.2 4x4 亮度块的帧内预测编码方式

当编码的宏块使用 4x4 亮度块帧内编码方式时，对于宏块中的每一个 4x4 亮度块有 9 种可能的预测方式。预测的 4x4 块由相邻的像素值（A~M）预测，如图 8.17 所示。

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

图 8.17 帧内预测时的像素分布

在解码器端，根据相关句法元素的值确定当前块采用的帧内预测方式。下文将进一步详细描述各种预测方式的特点。

#### 8.8.2.1 Intra\_4x4\_Vertical（模式 0）

在 Intra\_4x4\_Vertical 预测模式下，当前 4x4 块的像素值由相邻的像素 A、B、C 和 D 预测得出，如图 8.18 所示。

像素预测值由下式决定：

$$\text{pred4x4}_L[x, y] = p[x, -1], \quad x, y = 0..3 \tag{8.6}$$

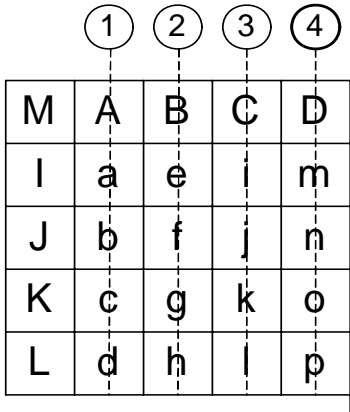


图 8.18 模式 0 帧内预测示意图

#### 8.8.2.2 Intra\_4x4\_Horizontal 预测模式（模式 1）

在 Intra\_4x4\_Horizontal 预测模式当前 4x4 块的像素值被相邻的像素 I、J、K 和 L 预测得出，如图 8.19 所示。

像素预测值由下式决定：

$$\text{pred4x4}_L[x, y] = p[-1, y], \quad x, y = 0..3 \quad (8.7)$$

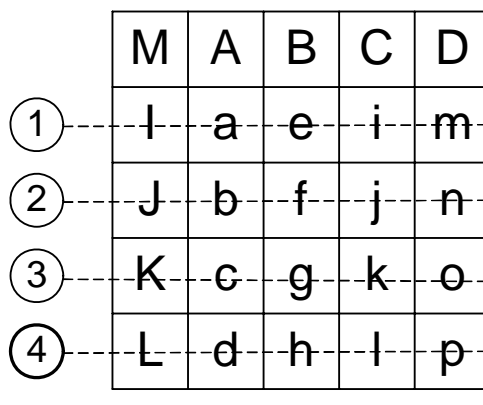


图 8.19 模式 1 帧内预测示意图

### 8.8.2.3 Intra\_4x4\_Diagonal\_Down\_Left 预测模式（模式 3）

在 Intra\_4x4\_Diagonal\_Down\_Left 预测模式下，当前 4x4 块的像素值被相邻的像素 A、B、C、D、E、F、G 和 H 预测得出，如图 8.20 所示。

像素预测值由下式决定：

$$\text{pred4x4}_L[x, y] = (p[6, -1] + 3 * p[7, -1] + 2) \gg 2, \quad x, y = 3$$

$$\text{pred4x4}_L[x, y] = (p[x+y, -1] + 2 * p[x+y+1, -1] + p[x+y+2, -1] + 2) \gg 2, \quad x, y \text{ 为其它值} \quad (8.8)$$

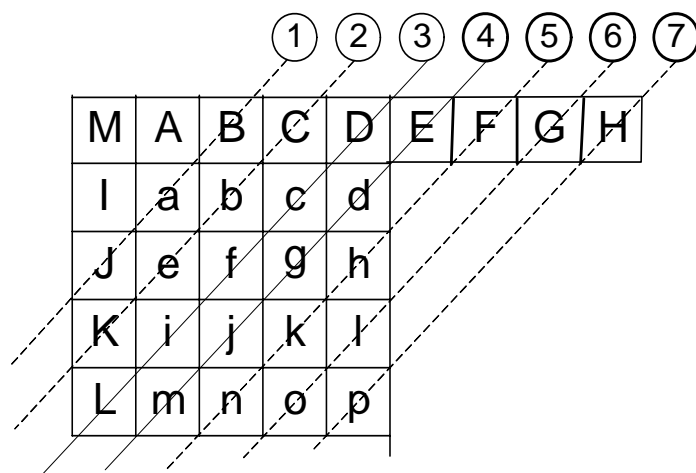


图 8.20 模式 3 帧内预测示意图

### 8.8.2.4 Intra\_4x4\_Diagonal\_Down\_Right 预测模式（模式 4）

在 Intra\_4x4\_Diagonal\_Down\_Right 预测模式下，当前 4x4 块的像素值被相邻的像素 A、B、C、D、I、J、K、L 和 M 预测得出，如图 8.21 所示。

像素预测值由下式决定：

$$\text{pred4x4}_L[x, y] = (p[x-y-2, -1] + 2 * p[x-y-1, -1] + p[x-y, -1] + 2) \gg 2, \quad x > y$$

$$\text{pred4x4}_L[x, y] = (p[-1, y-x-2] + 2 * p[-1, y-x-1] + p[-1, y-x] + 2) \gg 2, \quad x < y$$

$$\text{pred4x4}_L[x, y] = (p[0, -1] + 2 * p[-1, -1] + p[-1, 0] + 2) \gg 2, \quad x, y \text{ 为其它值} \quad (8.9)$$

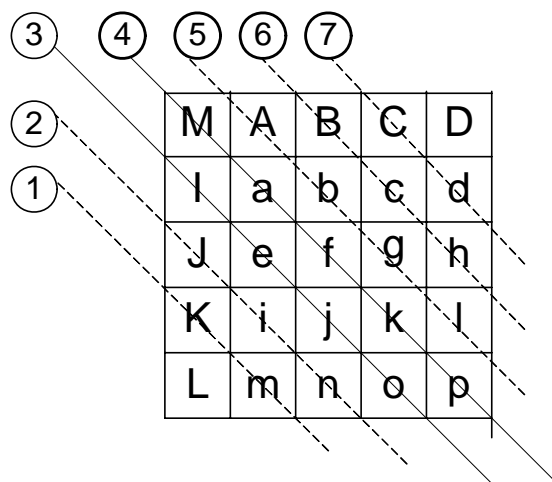


图 8.21 模式 4 帧间预测示意图

#### 8.8.2.5 Intra\_4x4\_Vertical\_Right 预测模式（模式 5）

在 Intra\_4x4\_Vertical\_Right 预测模式下，当前 4x4 块的像素值被相邻的像素 A、B、C、D、I、J、K、L 和 M 预测得出，如图 8.22 所示。

像素预测值由下式决定：

$\text{pred}_{4 \times 4_L}[x, y] = (p[x - (y \gg 1) - 1, -1] + p[x - (y \gg 1), -1] + 1) \gg 1, \quad 2 * x - y = 0, 2, 4, 6$

$\text{pred}_{4 \times 4_L}[x, y] = (p[x - (y \gg 1) - 2, -1] + 2 * p[x - (y \gg 1) - 1, -1] + p[x - (y \gg 1), -1] + 2) \gg 2, \quad 2 * x - y = 1, 3, 5$

$\text{pred}_{4 \times 4_L}[x, y] = (p[-1, 0] + 2 * p[-1, -1] + p[0, -1] + 2) \gg 2, \quad 2 * x - y = -1$

$\text{pred}_{4 \times 4_L}[x, y] = (p[-1, y - 1] + 2 * p[-1, y - 2] + p[-1, y - 3] + 2) \gg 2, \quad x, y \text{ 为其它值}$   
(8.10)

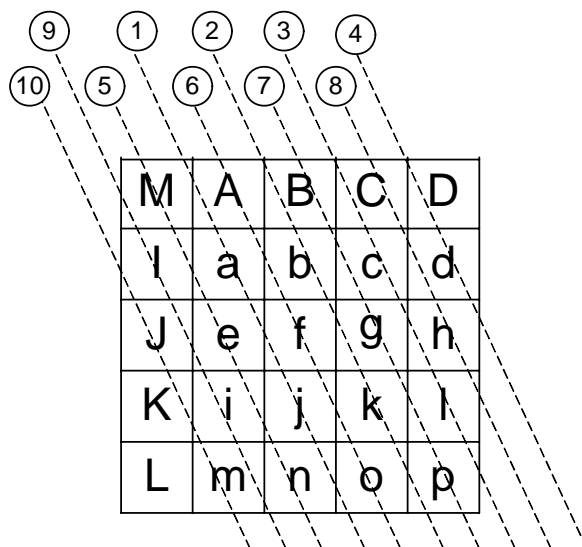


图 8.22 模式 5 帧内预测示意图

### 8.8.2.6 Intra\_4x4\_Horizontal\_Down 预测模式（模式 6）

在 Intra\_4x4\_Horizontal\_Down 预测模式下，当前 4x4 块的像素值被相邻的像素 A、B、C、I、J、K 和 L 预测得出，如图 8.23 所示。

像素预测值由下式决定：

$$\text{pred4x4}_L[x, y] = (p[-1, y - (x \gg 1) - 1] + p[-1, y - (x \gg 1)] + 1) \gg 1, \quad 2 * y - x = 0, 2, 4, 6$$

$$\text{pred4x4}_L[x, y] = (p[-1, y - (x \gg 1) - 2] + 2 * p[-1, y - (x \gg 1) - 1] + p[-1, y - (x \gg 1)] + 2) \gg 2, \quad 2 * y - x = 1, 3, 5$$

$$\text{pred4x4}_L[x, y] = (p[-1, 0] + 2 * p[-1, -1] + p[0, -1] + 2) \gg 2, \quad 2 * y - x = -1$$

$$\text{pred4x4}_L[x, y] = (p[x - 1, -1] + 2 * p[x - 2, -1] + p[x - 3, -1] + 2) \gg 2, \quad x, y \text{ 为其它值}$$

(8.11)

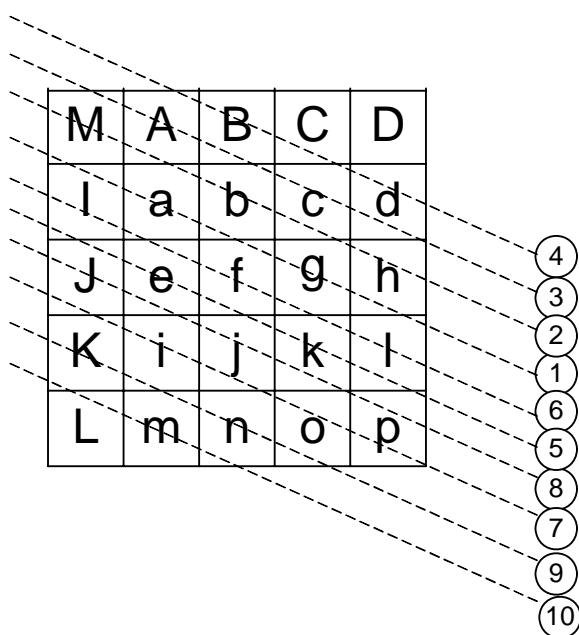


图 8.23 模式 6 帧内预测示意图

### 8.8.2.7 Intra\_4x4\_Vertical\_Left 预测模式（模式 7）

在 Intra\_4x4\_Vertical\_Left 预测模式下，当前 4x4 块的像素值被相邻的像素 A、B、C、D、E、F 和 G 预测得出，如图 8.24 所示。

像素预测值由下式决定：

$$\text{pred4x4}_L[x, y] = (p[x + (y \gg 1), -1] + p[x + (y \gg 1) + 1, -1] + 1) \gg 1, \quad y = 0, 2$$

$$\text{pred4x4}_L[x, y] = (p[x + (y \gg 1), -1] + 2 * p[x + (y \gg 1) + 1, -1] + p[x + (y \gg 1) + 2, -1] + 2) \gg 2, \quad y = 1, 3$$

(8.12)

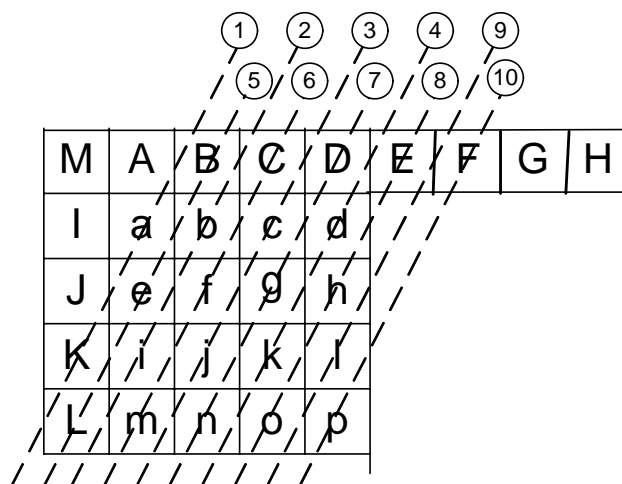


图 8.24 模式 7 帧内预测示意图

#### 8.8.2.8 Intra\_4x4\_Horizontal\_Up 预测模式（模式 8）

在 Intra\_4x4\_Horizontal\_Up 预测模式下，当前 4x4 块的像素值被相邻的像素 I、J、K 和 L 预测得出，如图 8.25 所示。

像素预测值由下式决定：

$\text{pred}_{4 \times 4_L}[x, y] = (p[-1, y + (x \gg 1)] + p[-1, y + (x \gg 1) + 1] + 1) \gg 1, x+2*y=0,2,4$

$\text{pred}_{4 \times 4_L}[x, y] = (p[-1, y + (x \gg 1)] + 2 * p[-1, y + (x \gg 1) + 1] + p[-1, y + (x \gg 1) + 2] + 2) \gg 2, x+2*y=1,3$

$\text{pred}_{4 \times 4_L}[x, y] = (p[-1, 2] + 3 * p[-1, 3] + 2) \gg 2, x+2*y=5$

$\text{pred}_{4 \times 4_L}[x, y] = p[-1, 3], x, y$  为其它值 (8.13)

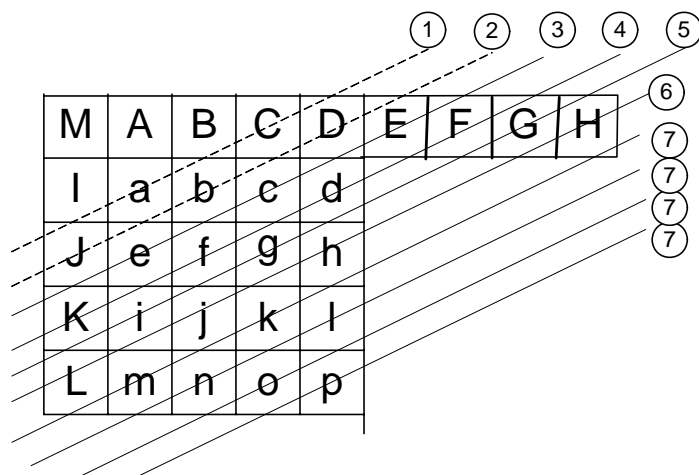


图 8.25 模式 8 帧内预测示意图

#### 8.8.2.9 Intra\_4x4\_DC 预测模式（直流预测）

在 Intra\_4x4\_DC 模式下在这种预测方式下，当相邻像素 A、B、C、D、I、J、K 及 L 均存在时

由其像素值的平均值预测当前 4x4 的亮度块；若只有像素 I~L 存在，则当前 4x4 块由 I~L 像素值的平均值预测得到；若只有像素 A~D 存在，则当前 4x4 块由 A~D 像素值的平均值预测得到；否则当前 4x4 块的预测值均为 128，如图 8.26 所示。

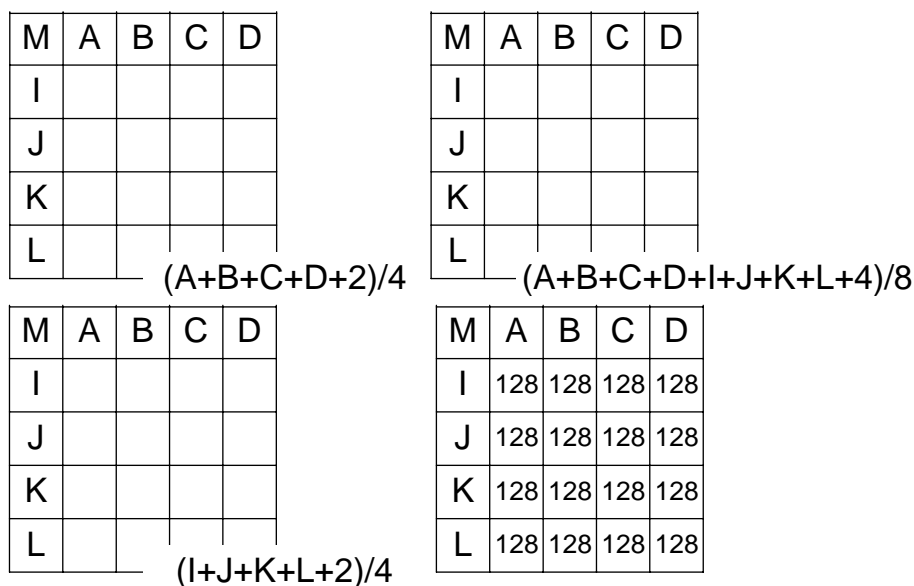


图 8.26 模式 2 直流预测示意图

### 8.8.3 16x16 亮度块的帧内预测方式

在这种预测方式下，整个 16x16 的亮度宏块同时被预测，且共有 4 中不同的预测方式，如表 8.3 所示。

表 8.3 16x16 亮度块的预测方式

预测模式序号	预测模式
0	Intra_16x16_Vertical
1	Intra_16x16_Horizontal
2	Intra_16x16_DC
3	Intra_16x16_Plane

#### 8.8.3.1 Intra\_16x16\_Vertical 预测模式（模式 0）

在 Intra\_16x16\_Vertical 预测模式下，当前宏块的所有像素由上方的像素 H 预测得到，如图 8.27 所示。

像素预测值由下式决定：

$$\text{pred}_L[x, y] = p[x, -1], \quad x, y = 0..15 \quad (8.14)$$



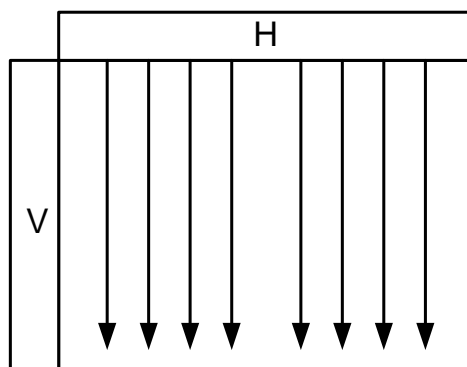


图 8.27 模式 0 帧内预测示意图

### 8.8.3.2. Intra\_16x16\_Horizontal 预测模式（模式 1）

在 Intra\_16x16\_Horizontal 预测模式下,当前宏块的所有像素由左方的像素 V 预测得到,如图 8.28 所示。

像素预测值由下式决定:

$$\text{pred}_L[x, y] = p[-1, y], \quad x, y = 0..15 \quad (8.15)$$

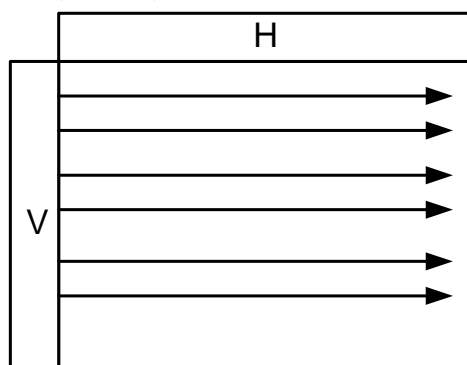


图 8.28 模式 1 帧内预测示意图

### 8.8.3.3 Intra\_16x16\_DC 预测模式（模式 2）

在 Intra\_16x16\_DC 预测模式下,当前宏块的所有像素由左方及上方的像素 V 及 H 的平均值预测得到,如图 8.29 所示。

若相邻像素 H 与 V 均存在,当前宏块的所有像素值为 (H+V) 的平均值;若只有像素 H 存在,当前宏块的所有像素值为 H 的平均值;若只有像素 V 存在,当前宏块的所有像素值为像素 V 的平均值;否则(像素 H 与 V 均不存在),当前宏块的所有像素值为 128。

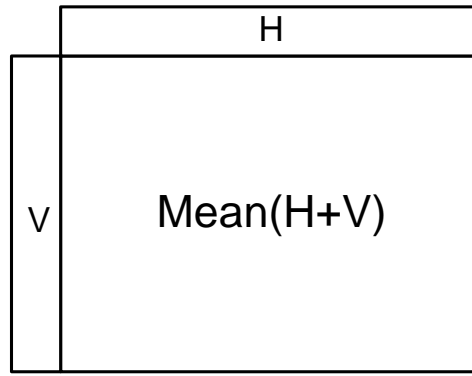


图 8.29 模式 2 直流预测示意图

#### 8.8.3.4 Intra\_16x16\_Plane 预测模式（模式 3）

在 Intra\_16x16\_Plane 预测模式下，如图 8.30 所示。

像素预测值由下式决定：

$$\text{pred}_L[x, y] = \text{Clip1}((a + b * (x - 7) + c * (y - 7) + 16) \gg 5), \quad (8.16)$$

在上式中：

$$a = 16 * (p[-1, 15] + p[15, -1])$$

$$b = (5 * H + 32) \gg 6$$

$$c = (5 * V + 32) \gg 6 \quad (8.17)$$

其中H，V由下式决定

$$H = \sum_{x'=0}^7 (x'+1) * (p[8+x', -1] - p[6-x', -1])$$

$$V = \sum_{y'=0}^7 (y'+1) * (p[-1, 8+y'] - p[-1, 6-y'])$$

(8.18)

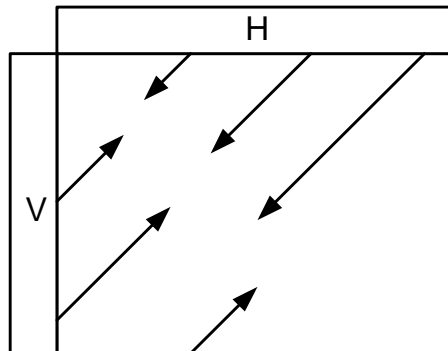


图 8.30 模式 3 帧内预测示意图

8.8.4 8x8 色度块的帧内预测方式

对于整个 8x8 的色度宏块，共有 4 种不同的预测方式，如表 8.4 所示。

表 8.4 色度块的预测方式

预测模式序号	预测模式
0	Intra_Chroma_DC
1	Intra_Chroma_Horizontal
2	Intra_Chroma_Vertical
3	Intra_Chroma_Plane

8.8.3.1 Intra\_Chroma\_Vertical I 预测模式（模式 0）

在 Intra\_Chroma\_Vertical 预测模式下，当前宏块的所有像素由上方的像素 H 预测得到，如图 8.31 所示。

像素预测值由下式决定：

$$pred_L[x, y] = p[x, -1], \quad x, y = 0..15$$

(8.19)

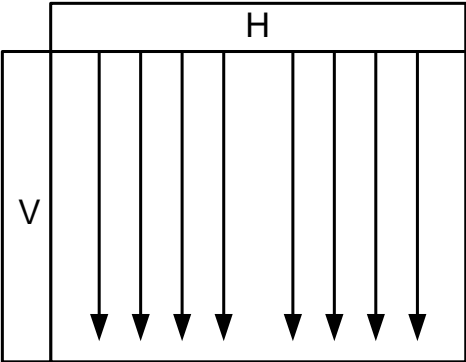


图 8.31 模式 0 帧内预测示意图

8.8.3.2.Intra\_Chroma\_Horizontal 预测模式（模式 1）

在 Intra\_Chroma\_Horizontal 预测模式下，当前宏块的所有像素由左方的像素 V 预测得到，如图 8.32 所示。

像素预测值由下式决定：

$$pred_L[x, y] = p[-1, y], \quad x, y = 0..15$$

(8.20)

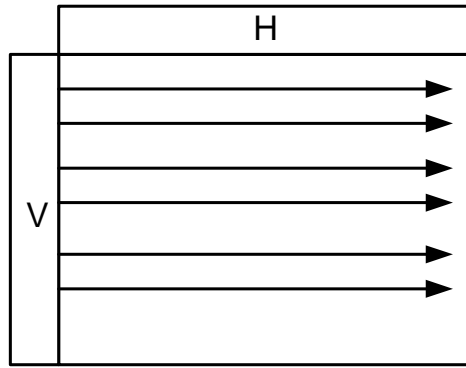


图 8.32 模式 1 帧内预测示意图

#### 8.8.3.3 Intra\_Chroma\_DC 预测模式（模式 2）

在 Intra\_Chroma\_DC 预测模式下，当前宏块的所有像素由左方及上方的像素 V 及 H 的平均值预测得到，如图 8.33 所示。

若相邻像素 H 与 V 均存在，当前宏块的所有像素值为 (H+V) 的平均值；若只有像素 H 存在，当前宏块的所有像素值为 H 的平均值；若只有像素 V 存在，当前宏块的所有像素值为像素 V 的平均值；否则（像素 H 与 V 均不存在），当前宏块的所有像素值为 128。

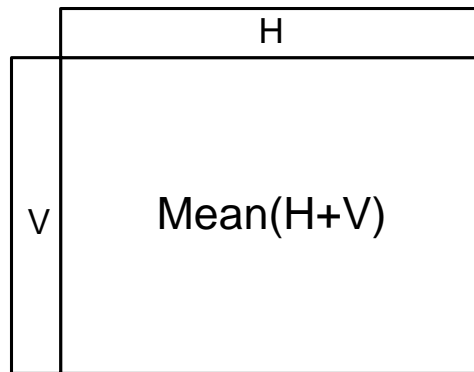


图 8.33 模式 2 直流预测示意图

#### 8.8.3.4 Intra\_Chroma\_Plane 预测模式（模式 3）

在 Intra\_Chroma\_Plane 预测模式下，如图 8.34 所示。

像素预测值由下式决定：

$$\text{pred}_L[x, y] = \text{Clip1}((a + b * (x - 7) + c * (y - 7) + 16) \gg 5), \quad (8.21)$$

在上式中：

$$\begin{aligned} a &= 16 * (p[-1, 15] + p[15, -1]) \\ b &= (5 * H + 32) \gg 6 \\ c &= (5 * V + 32) \gg 6 \end{aligned} \quad (8.22)$$

其中 H, V 由下式决定

$$H = \sum_{x'=0}^7 (x'+1) * (p[8+x', -1] - p[6-x', -1])$$

$$V = \sum_{y'=0}^7 (y'+1) * (p[-1, 8+y'] - p[-1, 6-y'])$$

(8.23)

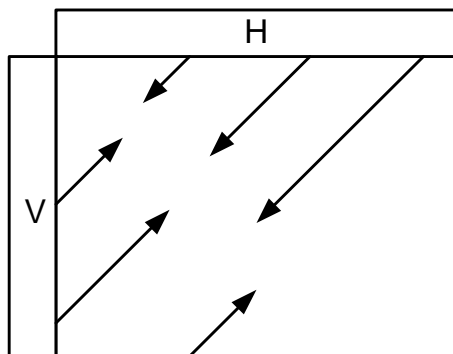


图 8.34 模式 3 帧内预测示意图

## 8.9 帧间预测解码处理

如第 6 章所述, H.264 采用树状结构的运动补偿技术, 提高了预测能力。特别是, 小块预测提高了模型处理更好的运动描述的能力, 产生更好的图像质量。H.264 运动向量的精度提高到 1/4 像素(亮度), 运动补偿算法的预测能力得到进一步提高。H.264 还提供多参考帧可选模式, 这将产生更好的视频质量和效率更高的视频编码。相对于 1 帧参考, 5 个参考帧可以节约 5%~10% 的比特率, 且有助于比特流的恢复。

在解码端, P 和 B 宏块编解码时需进行帧间预测解码处理, 其输出为帧间预测像素矩阵, 包括一个 16×16 的亮度点矩阵——pred<sub>L</sub> 和两个色度点 8×8 矩阵——pred<sub>C<sub>r</sub></sub> 和 pred<sub>C<sub>b</sub></sub>。其解码流程见图 8.35。

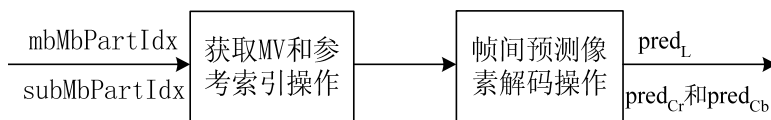


图 8.35 帧间预测解码总流程

其中, 获取 MV 分量及参考索引操作见 8.9.1, 其输入为 mbPartIdx, subMbPartIdx, 输出为 mvL0, mvL1, mvCL0, mvCL1 (亮度及色度 MV), refIdxL0, refIdxL1 (参考索引), predFlagL0, predFlagL1 (预测表使用标识); 帧间预测像素解码处理见 8.9.2, 输入为 mbPartIdx, subMbPartIdx, partWidth, partHeight, mvL0, mvL1, mvCL0, mvCL1, refIdxL0, refIdxL1, predFlagL0, predFlagL1; 输出为帧间预测像素, 一个 partWidth×partHeight 的亮度点矩阵 predPart<sub>L</sub> 和两个 (partWidth/2)×(partHeight/2) 色度点矩阵, predPart<sub>C<sub>r</sub></sub>, predPart<sub>C<sub>b</sub></sub>。

这里须说明的是帧间预测处理指 mbPartIdx 和 subMbPartIdx 指定的宏块或亚宏块的帧间预测处理。

简单一点的说, 预测宏块的形成就是将分割或者亚宏块分割预测像素放入宏块的正确位置, 形成矩阵。H.264 的帧间预测解码中, 宏块矩阵形成如下:

$$\text{pred}_L[xP + xS + x, yP + yS + y] = \text{predPart}_L[x, y];$$

$$x = 0 \dots \text{partWidth} - 1, y = 0 \dots \text{partHeight} - 1$$

(8.24)

$$\begin{aligned} \text{predc}[xP/2 + xS/2 + x, yP/2 + yS/2 + y] &= \text{predPartc}[x, y]; \\ x &= 0 \dots \text{partWidth}/2 - 1, y = 0 \dots \text{partHeight}/2 - 1 \end{aligned}$$

(8.25)

其中,  $(xP, yP)$  为宏块分割的左上像素相对宏块左上像素的位置。 $(xS, yS)$  为宏块亚分割的左上像素相对宏块分割的左上像素的位置。

本节只对 H.264 帧间预测解码处理的原理和流程作一定的介绍, 其具体操作, 如各部分推导请参考 H.264 Draft G050。

相关符号说明见第 7 章所示。

### 8.9.1 MV 分量及参考索引获取

该步骤输入为  $\text{mbPartIdx}$  和  $\text{subMbPartIdx}$ ; 输出为  $\text{mvL0}$ ,  $\text{mvL1}$ ,  $\text{mvCL0}$ ,  $\text{mvCL1}$ ,  $\text{refIdxL0}$ ,  $\text{refIdxL1}$ ,  $\text{predFlagL0}$ ,  $\text{predFlagL1}$ 。如图 8.36 所示。

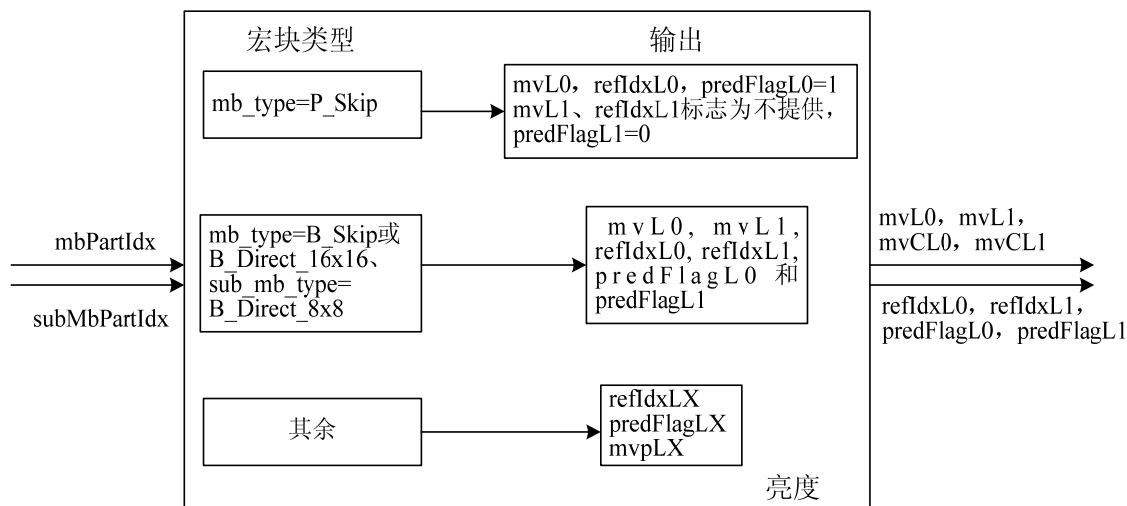


图 8.36 亮度 MV 及参考索引获取

其具体处理过程见下文。色度 MV 获取 8.9.1.4, 以  $\text{mvLX}$  and  $\text{refIdxLX}$  为输入,  $\text{mvCLX}$  为输出。本文先讲通常情况下的亮度 MV 如何获取, 再讨论特殊情况。最后讨论色度 MV 如何获得。

#### 8.9.1.3 亮度 MV 预测获取

当宏块类型为常用类型时, 亮度 MV 的获取如该节所示。该处理输入为  $\text{mbPartIdx}$ 、 $\text{subMbPartIdx}$ 、 $\text{LX}$ 、 $\text{refIdxLX}$ , 输出为运动矢量  $\text{mvLX}$  的预测  $\text{mvpLX}$ 。

$$\text{mvLX}[0] = \text{mvpLX}[0] + \text{mvd\_IX}[\text{mbPartIdx}][\text{subMbPartIdx}][0]$$

(8.26)

$$\text{mvLX}[1] = \text{mvpLX}[1] + \text{mvd\_IX}[\text{mbPartIdx}][\text{subMbPartIdx}][1]$$

(8.27)

该操作流程如图 8.37 所示。

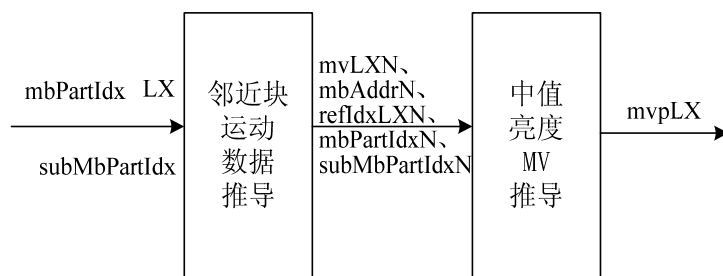


图 8.37 亮度 MV 推导流程

### 8.9.1.2 P 片、SP 片中跳跃宏块亮度 MV 获取

该处理用于宏块类型为 P\_Skip 时解码。其输出为 mvL0 和 refIdxL0（等于 0）。

此时，mvL0 推导如下：

第一步：根据上一节的邻近运动数据推导，以 mbPartIdx = 0, subMbPartIdx = 0, list suffix L0 为输入，输出为 mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, 和 refIdxL0B。

第二步：如下列任一命题为真，mvL0 的两个分量都置 0：

- ◆ mbAddrA 不提供；
- ◆ mbAddrB 不提供；
- ◆ refIdxL0A = 0 且 mvL0A 两个分量都等于 0；
- ◆ refIdxL0B = 0 且 mvL0B 两个分量都等于 0。

否则，根据上一节的中值亮度 MV 推导，以 mbPartIdx = 0, subMbPartIdx = 0, refIdxL0, and list suffix L0 为输入，mvL0 为输出。

注：这里输出直接为 mvL0，预测 MV 等于实际 MV。

### 8.9.1.3 B\_skip, B\_Direct\_16×16, B\_Direct\_8×8 的亮度 MV 获取

该处理用于宏块类型为 B\_Skip 或 B\_Direct\_16x16，或者亚宏块类型为 B\_Direct\_8x8 时的解码。其输入为 mbPartIdx 和 subMbPartIdx；输出为 refIdxL0, refIdxL1, mvL0 and mvL1, predFlagL0 and predFlagL1。

该处理还取决于 direct\_spatial\_mv\_pred\_flag 的值。direct\_spatial\_mv\_pred\_flag = 1 时，该处理输出模式为空间直接预测模式；direct\_spatial\_mv\_pred\_flag = 0 时，该处理输出模式为时间直接预测模式。详细过程请参考 H.264 Draft G050。

### 8.9.1.4 色度 MV 获取

该处理过程输入为亮度矢量 mvLX 和 refIdxLX，输出为色度矢量 mvCLX。色度 MV 是根据相应亮度 MV 推导而得。因为亮度 MV 精度为 1/4 像素，而色度精度为其一半，应为 1/8 像素精度。例如，亮度矢量指定 8×16 亮度像素时，相应色度矢量应针对 4×8 色度像素。

为了推出 mvCLX，有如下操作：

1) 当前宏块为帧宏块时，色度运动矢量 mvCLX 的水平和垂直分量通过相应亮度 mvLX 分量乘以 2 推出，该过程通过将 1/4 像素 mvLX 单元映射到 1/8 像素 mvCLX 单元实现。





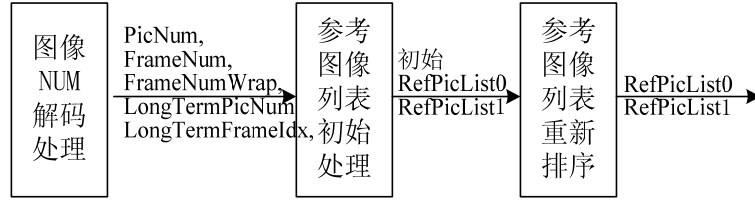


图 8.39 RefPicListX 推导

### 8.9.2.2 分像素内插处理

该处理过程输入为mbPartIdx、subMbPartIdx、亮度像素单元的partWidth和partHeight、mvLX、mvCLX、refPicLXL、refPicLXCb、refPicLXCc；输出为一个预测亮度像素值矩阵 (partWidth)×(partHeight)的 predPartLXL、两个预测色度像素值矩阵 (partWidth/2)×(partHeight/2)的 predPartLXCb, and predPartLXCc。

这里，设( xAL, yAL )为当前分割左上亮度像素整像素单元位置，这是相对给出亮度像素二维矩阵左上像素位置而言。设( xIntL, yIntL )为整像素单元亮度位置，( xFracL, yFracL )为 1/4 像素单元偏移。

$$xInt_L = xA_L + (mvLX[0] \gg 2) + x_L \quad (8.30)$$

$$yInt_L = yA_L + (mvLX[1] \gg 2) + y_L \quad (8.31)$$

$$xFrac_L = mvLX[0] \& 3 \quad (8.32)$$

$$yFrac_L = mvLX[1] \& 3 \quad (8.33)$$

predLXL[ xL, yL ]推导见8.9.2.2.1，以 ( xIntL, yIntL )、( xFracL, yFracL )、refPicLXL 为输入。

同样，设( xIntC, yIntC )为整像素单元亮度位置，( xFracC, yFracC )为1/8像素单元偏移。

$$xInt_C = (xA_L \gg 1) + (mvCLX[0] \gg 3) + x_C \quad (8.34)$$

$$yInt_C = (yA_L \gg 1) + (mvCLX[1] \gg 3) + y_C \quad (8.35)$$

$$xFrac_C = mvCLX[0] \& 7 \quad (8.36)$$

$$yFrac_C = mvCLX[1] \& 7 \quad (8.37)$$

predPartLXCb[ xC, yC ]和predPartLXCc[ xC, yC ]推导见2)，以( xIntC, yIntC )和( xFracC, yFracC )及 refPicLXCb、refPicLXCc 为输入。

#### 1) 亮度像素内插处理

该处理过程输入为亮度整像素位置( xIntL, yIntL )、亮度分像素位置偏移( xFracL, yFracL )、参考图像refPicLXL的亮度像素矩阵。输出为预测亮度像素值predPartLXL[ xL, yL ]。

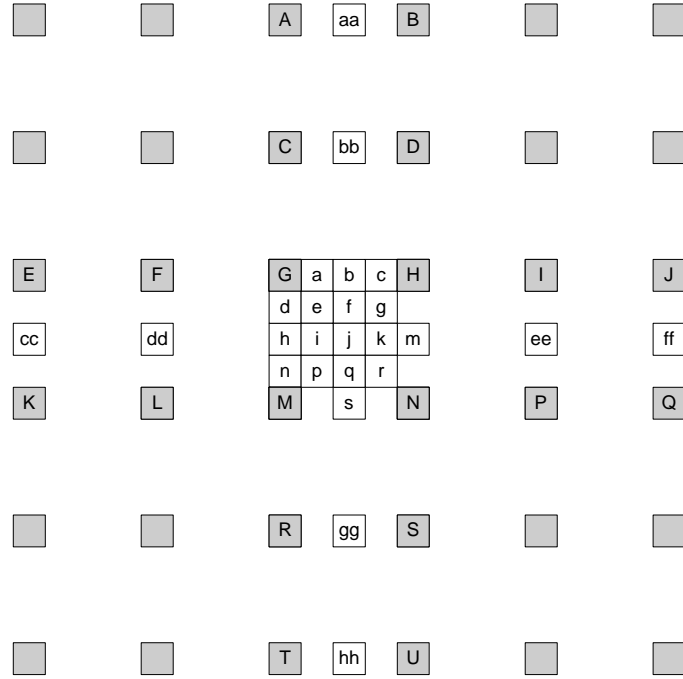


图8.40 1/4亮度像素内插

在图 8.40 中，标有大些字母阴影块位置为参考亮度像素值的二维矩阵的整数位置。这些像素用来推导  $\text{predPartLX}_L[x_L, y_L]$ 。

参考亮度像素矩阵  $\text{refPicLX}_L$  中，各像素位置  $(xZ_L, yZ_L)$  推导如下，Z 可为 A, B, C, D, E, F, G, H, I, J, K, L, M, N, P, Q, R, S, T, 或 U:

$$\begin{aligned} xZ_L &= \text{Clip3}(0, \text{PicWidthInSamples}_L - 1, x\text{Int}_L + xDZ_L) \\ yZ_L &= \text{Clip3}(0, \text{PicHeightInSamples}_L - 1, y\text{Int}_L + yDZ_L) \end{aligned} \quad (8.38)$$

表8.6给出了相应亮度像素位置

表8.6 亮度整像素位置

Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P	Q	R	S	T	U
$xDZ_L$	0	1	0	1	-2	-1	0	1	2	3	-2	-1	0	1	2	3	0	1	0	1
$yDZ_L$	-2	-2	-1	-1	0	0	0	0	0	0	1	1	1	1	1	1	2	2	3	3

在给出整像素位置  $(xA_L, yA_L)$  到  $(xU_L, yU_L)$  像素后，分像素位置“a”到“s”的像素值便可计算出来。其中，半像素通过 6 抽头滤波器  $(1, -5, 20, 20, -5, 1)$  实现，1/4 像素位置像素则通过整像素和半像素平均而得，具体如下。

(1) 计算半像素位置像素b和h，先计算中间量 $b_1$ 和 $h_1$ ，再计算b和h。

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (8.39)$$

$$h_1 = (A - 5 * C + 20 * G + 20 * M - 5 * R + T) \quad (8.40)$$

$$b = \text{Clip1}((b_1 + 16) \gg 5) \quad (8.41)$$

$$h = \text{Clip1}((h_1 + 16) \gg 5) \quad (8.42)$$

(2) 计算半像素位置像素j，同样先计算中间量 $j_1$ ， $j_1$ 通过对水平或垂直方向最近半像素滤波而得。

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff, \text{ 或者} \quad (8.43)$$

$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8.44)$$

这里，中间量aa、bb、gg、 $s_1$ 、hh、cc、dd、ee、 $m_1$ 、ff类似与 $b_1$ 和 $h_1$ 方式获得。最终，

$$j = \text{Clip1}((j_1 + 512) \gg 10) \quad (8.45)$$

(3) s和m 类似于b 和h,方法获得,

$$s = \text{Clip1}((s_1 + 16) \gg 5) \quad (8.46)$$

$$m = \text{Clip1}((m_1 + 16) \gg 5) \quad (8.47)$$

(4) 1/4像素a、c、d、n、f、i、k、q由最近的整像素和半像素平均得出,

$$a = (G + b + 1) \gg 1 \quad (8.48)$$

$$c = (H + b + 1) \gg 1 \quad (8.49)$$

$$d = (G + h + 1) \gg 1 \quad (8.50)$$

$$n = (M + h + 1) \gg 1 \quad (8.51)$$

$$f = (b + j + 1) \gg 1 \quad (8.52)$$

$$i = (h + j + 1) \gg 1 \quad (8.53)$$

$$k = (j + m + 1) \gg 1 \quad (8.54)$$

$$q = (j + s + 1) \gg 1. \quad (8.55)$$

(4) 1/4像素e、g、p、r 由两个对角线上的半像素获得,

$$e = (b + h + 1) \gg 1 \quad (8.56)$$

$$g = (b + m + 1) \gg 1 \quad (8.57)$$

$$p = (h + s + 1) \gg 1 \quad (8.58)$$

$$r = (m + s + 1) \gg 1. \quad (8.59)$$

分像素单元( $x\text{Frac}_L, y\text{Frac}_L$ )中的亮度位置偏移指定了产生的整像素和分像素中为预测亮度像素值  $\text{predPartLXL}[x_L, y_L]$  的像素, 如表 8.7 所示。

表8.7 亮度预测像素 $\text{predPartLXL}[x_L, y_L]$ 分配

$x\text{Frac}_L$	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
$y\text{Frac}_L$	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
$\text{predPartLXL}[x_L, y_L]$	G	d	h	n	a	e	i	p	b	f	j	q	c	g	k	r

## 2) 色度像素内插处理

该处理过程输入为一色度整像素位置( $x\text{Int}_C, y\text{Int}_C$ )、一色度分像素位置 ( $x\text{Frac}_C, y\text{Frac}_C$ )及参考图像 $\text{refPicLXC}$ 的色度像素。输出为 $\text{predPartLXC}[x_C, y_C]$ 。

图 8.41 中, A、B、C、D 表示色度像素二维矩阵  $\text{refPicLXC}$  的整像素位置像素。

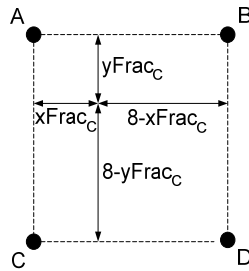


图8.41 色度分像素内插

这些像素用来产生  $\text{predPartLXC}[x_C, y_C]$ 。如下,

$$xA_C = \text{Clip3}(0, \text{PicWidthInSamples}_C - 1, x\text{Int}_C) \quad (8.60)$$

$$xB_C = \text{Clip3}(0, \text{PicWidthInSamples}_C - 1, x\text{Int}_C + 1) \quad (8.61)$$

$$xC_C = \text{Clip3}(0, \text{PicWidthInSamples}_C - 1, x\text{Int}_C) \quad (8.62)$$

$$xD_C = \text{Clip3}(0, \text{PicWidthInSamples}_C - 1, x\text{Int}_C + 1) \quad (8.63)$$

$$yA_C = \text{Clip3}(0, \text{PicHeightInSamples}_C - 1, y\text{Int}_C) \quad (8.64)$$

$$yB_C = \text{Clip3}(0, \text{PicHeightInSamples}_C - 1, y\text{Int}_C) \quad (8.65)$$

$$yC_c = \text{Clip3}(0, \text{PicHeightInSamples}_c - 1, yInt_c + 1) \quad (8.66)$$

$$yD_c = \text{Clip3}(0, \text{PicHeightInSamples}_c - 1, yInt_c + 1) \quad (8.67)$$

预测色度像素值  $\text{predPartLX}_c[x_c, y_c]$  推导如下:

$$\text{predPartLX}_c[x_c, y_c] = ((8 - xFrac_c) * (8 - yFrac_c) * A + xFrac_c * (8 - yFrac_c) * B + (8 - xFrac_c) * yFrac_c * C + xFrac_c * yFrac_c * D + 32) >> 6 \quad (8.68)$$

### 8.9.2.3 加权像素预测处理

该处理过程输入为  $\text{mbPartIdx}$ 、 $\text{subMbPartIdx}$ 、 $\text{predFlagL0}$  和  $\text{predFlagL1}$ 、 $\text{predPartLX}_L$ 、 $\text{predPartLX}_{Cb}$  和  $\text{predPartLX}_{Cr}$ 。输出为  $\text{predPart}_L$ 、 $\text{predPart}_{Cb}$  和  $\text{predPart}_{Cr}$ 。

P 片和 SP 片中  $\text{predFlagL0}=1$  宏块或分割有如下处理过程:

- 1) 若  $\text{weighted\_pred\_flag}=0$ , 采用缺省加权处理, 见 1)。
- 2) 若  $\text{weighted\_pred\_flag}=1$ , 采用加权处理, 见 2)。

B 片中  $\text{predFlagL0}=1$  或  $\text{predFlagL}=1$  宏块或分割有如下处理过程:

- 1) 若  $\text{weighted\_bipred\_idc}=0$ , 采用缺省加权处理, 见 1)。
- 2) 若  $\text{weighted\_bipred\_idc}=1$ , 采用加权处理, 见 2)。
- 3) 若  $\text{weighted\_bipred\_idc}=2$ , 如下:
  - $\text{predFlagL0}=1$  且  $\text{predFlagL1}=1$ , 采用加权处理, 见 2)。
  - $\text{predFlagL0}=1$  或  $\text{predFlagL1}=1$ , 采用缺省加权处理, 见 1)。

#### 1) 缺省加权像素预测处理

根据预测方块类型, 有以下约定:

- 1) 推导亮度  $\text{predPart}_L[x, y]$  时, C 设为 L, x 为  $0 \sim \text{partWidth} - 1$ , y 为  $0 \sim \text{partHeight} - 1$ 。
- 2) 推导色度 Cb  $\text{predPart}_{Cb}[x, y]$  时, C 设为 Cb, x 为  $0 \sim \text{partWidth} / 2 - 1$ , y 为  $0 \sim \text{partHeight} / 2 - 1$ 。
- 3) 推导色度 Cr  $\text{predPart}_{Cr}[x, y]$  时, C 设为 Cb, x 为  $0 \sim \text{partWidth} / 2 - 1$ , y 为  $0 \sim \text{partHeight} / 2 - 1$ 。

则, 缺省加权情况下的预测值如下:

$$1) \text{predFlagL0}=1 \text{ 且 } \text{predFlagL1}=0 \text{ 时,} \quad \text{predPart}_c[x, y] = \text{predPartL0}_c[x, y] \quad (8.69)$$

$$2) \text{predFlagL0}=0 \text{ 且 } \text{predFlagL1}=1 \text{ 时,} \quad \text{predPart}_c[x, y] = \text{predPartL1}_c[x, y] \quad (8.70)$$

$$3) \text{predFlagL0}=1 \text{ 且 } \text{predFlagL1}=1 \text{ 时,} \quad \text{predPart}_c[x, y] = (\text{predPartL0}_c[x, y] + \text{predPartL1}_c[x, y] + 1) >> 1. \quad (8.71)$$

#### 2) 加权像素预测处理

根据预测方块类型, 有以下约定:

- 1) 推导亮度  $\text{predPart}_L[x, y]$  时, C 设为 L, x 为  $0 \sim \text{partWidth} - 1$ , y 为  $0 \sim \text{partHeight} - 1$ 。
- 2) 推导色度 Cb  $\text{predPart}_{Cb}[x, y]$  时, C 设为 Cb, x 为  $0 \sim \text{partWidth} / 2 - 1$ , y 为  $0 \sim \text{partHeight} / 2 - 1$ 。

3) 推导色度Cr  $\text{predPartCr}[x, y]$ 时, C 设为Cb, x为 $0 \sim \text{partWidth} / 2 - 1$ , y为  $0 \sim \text{partHeight} / 2 - 1$ 。

预测值如下:

1) 分割  $\text{mbPartIdx} \backslash \text{subMbPartIdx}$  的  $\text{predFlagL0}=1$  且  $\text{predFlagL1}=0$  时,

```
if( logWD >= 1 )
    predPartC[ x, y ] = Clip1( ( ( predPartL0C[ x, y ] * w0 + 2logWD - 1 ) >> logWD ) + o0 )
else
    (8.72)
```

```
    predPartC[ x, y ] = Clip1( predPartL0C[ x, y ] * w0 + o0 )
```

2) 分割  $\text{mbPartIdx} \backslash \text{subMbPartIdx}$  的  $\text{predFlagL0}=0$  且  $\text{predFlagL1}=1$  时,

```
if( logWD >= 1 )
    predPartC[ x, y ] = Clip1( ( ( predPartL1C[ x, y ] * w1 + 2logWD - 1 ) >> logWD ) + o1 )
else
    (8.73)
```

```
    predPartC[ x, y ] = Clip1( predPartL1C[ x, y ] * w1 + o1 )
```

3) 分割  $\text{mbPartIdx} \backslash \text{subMbPartIdx}$  的  $\text{predFlagL0}=1$  且  $\text{predFlagL1}=1$  时,

```
predPartC[ x, y ] = Clip1( ( ( predPartL0C[ x, y ] * w0 + predPartL1C[ x, y ] * w1 + 2logWD ) >>
    ( logWD + 1 ) ) + ( ( o0 + o1 + 1 ) >> 1 ) )
(8.74)
```

上述推导过程中变量推导如下:

1) 若  $\text{weighted\_bipred\_idc}=2$  且  $\text{slice\_type}=B$  时,

```
logWD = 5
o0 = 0
o1 = 0
(8.75)
```

$w0$  和  $w1$  推导如下: .

· $\text{DiffPicOrderCnt}(\text{picA}, \text{picB})=0$  或一个或两个参考图像为长期参考图像或 $(\text{DistScaleFactor} \gg$

2)  $< -64$  或 $(\text{DistScaleFactor} \gg 2) > 128$  时,

```
w0 = 32
w1 = 32
(8.76)
```

·否则,

```
w0 = 64 - (DistScaleFactor >> 2)
w1 = DistScaleFactor >> 2
(8.77)
```

2)若 P 片或 SP 片  $\text{weighted\_pred\_flag}=1$  或 B 片  $\text{weighted\_bipred\_idc}=1$  时,

$\text{refIdxL0WP}$  和  $\text{refIdxL1WP}$  推导如下:

· $\text{MbaffFrameFlag}=1$  且当前宏块为场宏块时,

```
refIdxL0WP = refIdxL0 >> 1
(8.78)
refIdxL1WP = refIdxL1 >> 1
```

(8.79)

·MbaffFrameFlag=0 或当前宏块为帧宏块时,

refIdxL0WP = refIdxL0

(8.80)

refIdxL1WP = refIdxL1

(8.81)

logWD、w0、w1、o0、o1 推导如下:

·predPartC[ x, y ]中 C 为 L 时,

logWD = luma\_log2\_weight\_denom

w0 = luma\_weight\_l0[ refIdxL0WP ]

w1 = luma\_weight\_l1[ refIdxL1WP ]

o0 = luma\_offset\_l0[ refIdxL0WP ]

o1 = luma\_offset\_l1[ refIdxL1WP ]

(8.82)

·predPartC[ x, y ]中 C 为 Cb 或 Cr, iCbCr = 0, iCbCr = 1 时,

logWD = chroma\_log2\_weight\_denom

w0 = chroma\_weight\_l0[ refIdxL0WP ][ iCbCr ]

w1 = chroma\_weight\_l1[ refIdxL1WP ][ iCbCr ]

o0 = chroma\_offset\_l0[ refIdxL0WP ][ iCbCr ]

o1 = chroma\_offset\_l1[ refIdxL1WP ][ iCbCr ]

(8.83)

注: 在直接加权预测模式中, predFlagL0=1 和 predFlagL1=1 时, 必须遵守以下约束:

$-128 \leq w0 + w1 \leq 127$

(8.84)

而非直接模式中, 则满足  $-64 \leq w0, w1 \leq 128$  即可。

## 8.10 变换系数解码

H.264 中, 变换系数解码包括逆整数 DCT 及反量化。这个过程的输出与整个解码器的实际输出的差别在于图像样点位置不一致以及存在方块效应。本节介绍 H.264 中的变换系数解码和去方块滤波(去除方块效应)前的图像重建过程。本节解码器工作过程如图 8.42 所示, 图 8.43 显示其流程图。如果输入块是色度块或帧内 16×16 预测模式的亮度块, 则需要先对直流分量进行逆 Hadamard 变换, 恢复整数 DCT 变换的直流分量。

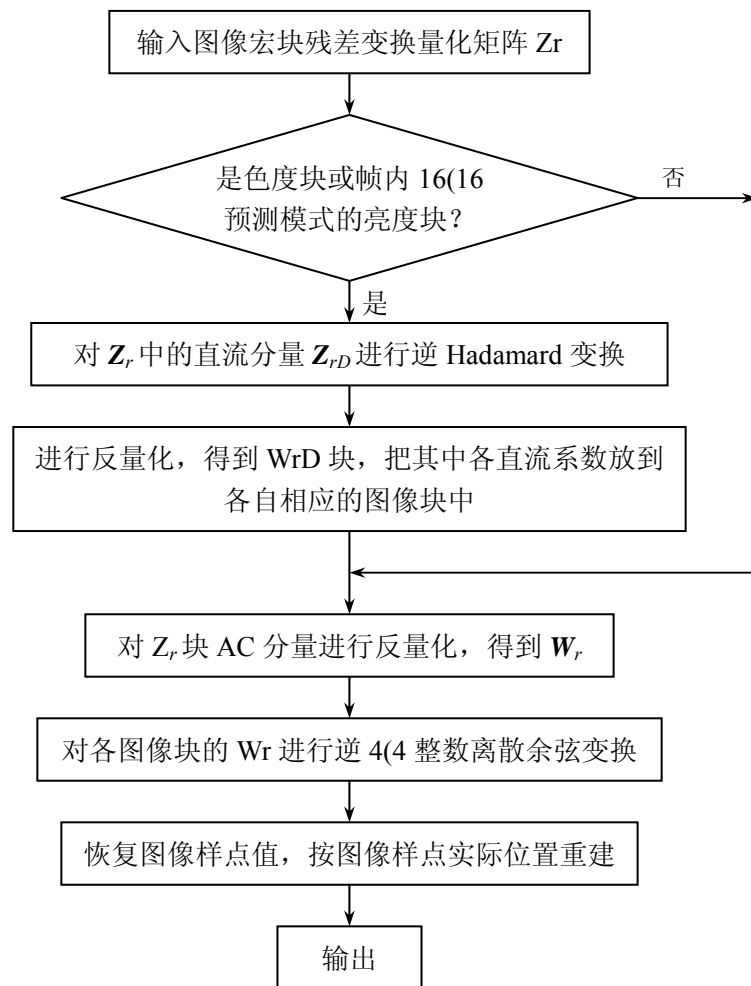


图 8.42 解码器中变换系数解码及图像重建过程

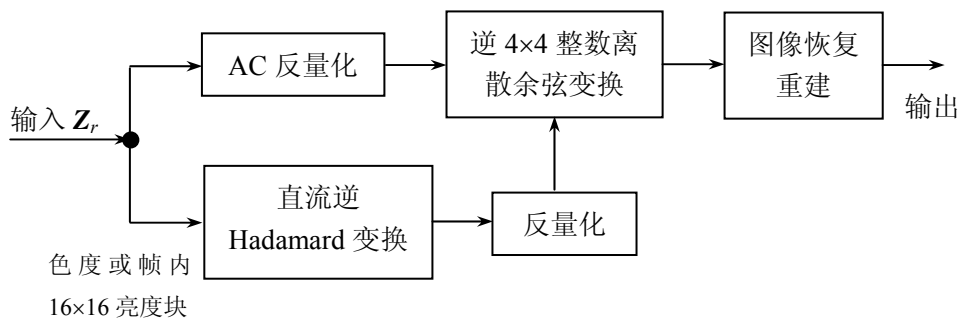


图 8.43 解码器中变换系数解码及图像重建流程图

### 8.10.1 变换系数逆扫描过程

H.264 多数场合是对  $4 \times 4$  图像块进行处理的, 在一个宏块中包含  $4 \times 4$  个图像块。往往需要对  $4 \times 4$  矩阵进行操作运算, 而在网络传输等应用中数据是以一维形式进行传递的。在编码器中将二维编码信息以扫描方式变成一维数据输出, 在解码器端则需要将这个一维数据转换成二维数组或矩阵进行

运算。本小节的功能是将输入的一个长为 16 的图像变换系数的序列转换成各图像块在宏块中的对应坐标位置，以一个 4×4 二维数组表示。

解码过程通过逆扫描过程将变换系数量化值序列映射到对应的坐标上。有两种逆扫描模式，如图 8.44 所示。逆 zig-zag 扫描应用于帧宏块，而逆场扫描应用于场宏块。

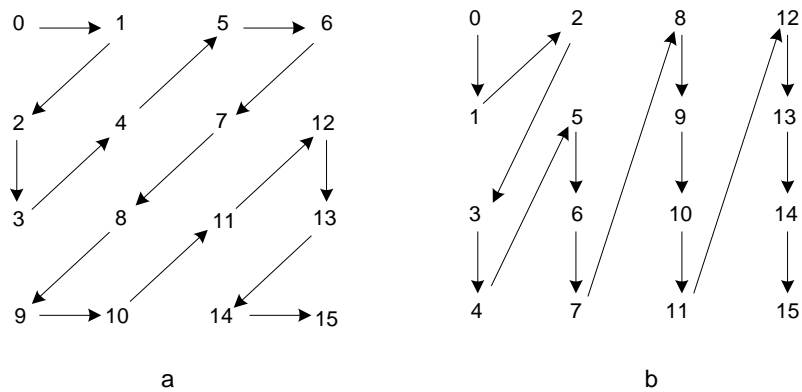


图 8.44 逆 a) Zig-zag 扫描；b) 逆场扫描

表 8.8 提供输入的 16 个元素数列的序号 *idx* 到输出的二维数组 *c* 的序号 *i* 和 *j* 的映射。

表 8.8 输入数列序号*idx*与输出二维数组元素*cij* 的映射关系

<i>idx</i>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
逆 zig-zag	<i>c</i> <sub>00</sub>	<i>c</i> <sub>01</sub>	<i>c</i> <sub>10</sub>	<i>c</i> <sub>20</sub>	<i>c</i> <sub>11</sub>	<i>c</i> <sub>02</sub>	<i>c</i> <sub>03</sub>	<i>c</i> <sub>12</sub>	<i>c</i> <sub>21</sub>	<i>c</i> <sub>30</sub>	<i>c</i> <sub>31</sub>	<i>c</i> <sub>22</sub>	<i>c</i> <sub>13</sub>	<i>c</i> <sub>23</sub>	<i>c</i> <sub>32</sub>	<i>c</i> <sub>33</sub>
逆场扫描	<i>c</i> <sub>00</sub>	<i>c</i> <sub>10</sub>	<i>c</i> <sub>01</sub>	<i>c</i> <sub>20</sub>	<i>c</i> <sub>30</sub>	<i>c</i> <sub>11</sub>	<i>c</i> <sub>21</sub>	<i>c</i> <sub>31</sub>	<i>c</i> <sub>02</sub>	<i>c</i> <sub>12</sub>	<i>c</i> <sub>22</sub>	<i>c</i> <sub>32</sub>	<i>c</i> <sub>03</sub>	<i>c</i> <sub>13</sub>	<i>c</i> <sub>23</sub>	<i>c</i> <sub>33</sub>

### 8.10.2 DCT 变换系数中直流系数的逆变换量化

如果当前处理的图像宏块是色度块或帧内 16×16 预测模式的亮度块，则需要先恢复各图像块的 DCT 变换的直流系数。

#### 8.10.2.1 帧内 16×16 预测模式宏块的亮度直流变换系数的比例变换

与一般设象的不同，这个过程是先进进行逆 **Hadamard** 变换，而不是反量化。它的输入是帧内 16×16 预测模式宏块的亮度 DC 变换系数量化值，表示为 4×4 数组 **Z<sub>rD</sub>**；输出是恢复的宏块中 4×4 个亮度块的 DCT 变换直流系数值，表示成一个 4×4 数组 **W<sub>rD</sub>**。

4×4 亮度直流变换系数的逆 **Hadamard** 变换定义为：

$$W_{QD} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} Z_{rD} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

(8.85)

类似于正向 **Hadamard** 变换和 DCT 变换，可以将式(8.85)的矩阵乘法运算改造成两次一维变换，例如先对量化矩阵的每行进行一维变换，然后对经行变换所得数据块的每列再应用一维变换。而每次一维变换可以采用蝶形快速算法，节省计算时间，如图 8.45 所示。



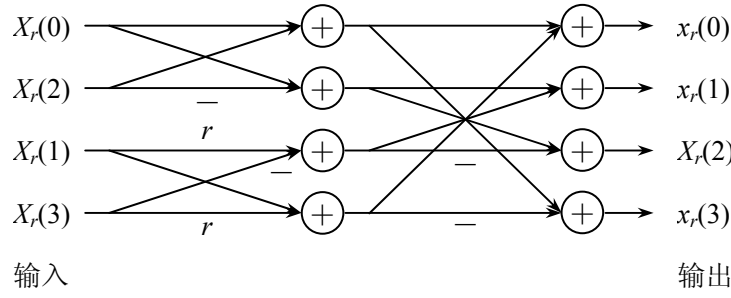


图 8.45 一维快速反变换算法

其中,  $r=1/2$ : 逆整数 DCT 变换;  $r=1$ : 逆 Hadamard 变换

反变换后, 根据亮度的量化参数  $QP$  进行反量化及逆 DCT 中的比例变换:

$$W_{rD(i,j)} = \begin{cases} (W_{QD(i,j)} V_{00}) \ll (QP/6 - 2) & QP \geq 12 \\ (W_{QD(i,j)} V_{00} + 2^{1-QP/6}) \gg (2 - QP/6) & QP < 12 \end{cases} \quad (i,j=0,\dots,3) \quad (8.86)$$

其中,  $V$  是融合在反量化过程中的逆 DCT 中的比例变换系数, 它的定义类似于  $MF$  系数,  $V_{00}$  是对应位置在  $(0, 0)$  上的  $V$  值。不同位置的  $V$  值如表 8.9 所示。

表8.9 H.264中V值

$QP$ \ 样点位置	$(0, 0), (2, 0), (2, 2), (0, 2)$	$(1, 1), (1, 3), (3, 1), (3, 3)$	其它样点位置
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

计算后要把相应得到的 DCT 直流系数放到对应图像亮度块的直流分量位置上。

### 8.10.2.2 色度直流变换系数的比例变换

这个过程与帧内  $16 \times 16$  预测模式宏块的亮度直流变换系数的比例变换类似, 它的输入是宏块色度直流变换系数量化值, 是一个  $2 \times 2$  数组  $Z_{rD}$ 。输出是 4 个反量化过的 DCT 中的直流系数值, 表示成  $2 \times 2$  数组  $W_{rD}$ 。

$2 \times 2$  色度直流变换系数的逆 Hadamard 变换定义为:

$$W_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} Z_{rD} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (8.87)$$

反变换后, 根据色度的量化系数  $QP_C$  进行反量化及逆 DCT 中的比例变换:

$$W_{rD(i,j)} = \begin{cases} (W_{QD(i,j)} V_{00}) \ll (QP_C/6 - 1) & QP_C \geq 6 \\ (W_{QD(i,j)} V_{00}) \gg 1 & QP_C < 6 \end{cases} \quad (i,j=0,\dots,3) \quad (8.88)$$

其中,  $QP_C$  的值由亮度的量化系数  $QP$  及 H.264 中定义的句法元素 `chroma_qp_index_offset` 决

定。 $QP_C$  值由表 8.10 根据  $qP_I$  值查找。 $qP_I$  值由下式计算：

$$qP_I = \text{Clip3}(0, 51, QP_Y + \text{chroma\_qp\_index\_offset}) \quad (8.89)$$

其中，函数 Clip3 为限幅函数，表示如果  $QP_Y + \text{chroma\_qp\_index\_offset}$  小于函数中第一个参数值 0，则  $qP_I$  为 0；如果  $QP_Y + \text{chroma\_qp\_index\_offset}$  大于函数中第二个参数值 51，则  $qP_I$  为 51；否则  $qP_I$  为实际的  $QP_Y + \text{chroma\_qp\_index\_offset}$  值。

表8.10 QPC值与qPI的对应关系

$qP_I$	<30	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
$QP_C$	$=qP_I$	29	30	31	32	32	33	34	34	35	35	36	36	37	37	37	38	38	38	39	39	39	39

计算后要把相应得到的 DCT 直流系数放到对应图像色度块的直流分量位置上。

### 8.10.3 残差变换系数的反量化

同量化过程一样，反量化过程也融合了逆 DCT 变换中的乘法运算。首先，如果当前处理的图像块是色度块或帧内 16×16 预测模式的亮度块，则反量化输出矩阵  $W_r$  中的直流系数  $W_{r(0,0)}$  直接为前节计算结果。对 DCT 交流分量或图像块不是前面情况的直流分量，按式(8.90)计算：

$$W_{r(i,j)} = (W_{r(i,j)} V_{(i,j)}) \ll (QP/6) \quad (i, j=0, \dots, 3) \quad (8.90)$$

其中， $QP$  为该块（亮度或色度）的量化参数，色度的量化参数的确定见上节。

### 8.10.4 残差变换系数的逆 DCT 变换

按照整数 DCT 变换的原理，逆 DCT 变换的公式应该为：

$$X_r = C_i^T (W \otimes E_i) C_i$$

$$= \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left( W \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}$$

(8.91)

其中，“ $\otimes E_i$ ”已在前面的反量化中完成，其结果为  $W_r$ ，所以这里只进行以下运算：

$$X_r = C_i^T W_r C_i \quad (8.92)$$

同样，式(8.92)也可以分成两次一维变换。

### 8.10.5 去方块滤波前的图像恢复与重建

到上节为至，计算结果是图像与预测值之前的残差值矩阵，实际输出的各图像样点值  $U_{ij}$ （ $i$  和  $j$  为图像样点在图像块中的行和列）应该是在该点上得到的残差值与前面的预测值之和，这样得到图像各样点的恢复值。

图像的重建是将上面恢复的图像样点值赋值到实际的图像输出矩阵中。设当前宏块亮度样本的左上角位置为  $(xP, yP)$ ，当前图像块在宏块中的位置为  $(xO, yO)$ ，恢复样本矩阵为  $U$ 。

如果  $U$  是亮度块，对  $4 \times 4$  亮度块中的每个样本元素  $U_{ij}$ ，根据变量  $MbaffFrameFlag$  进行以下操作：

① 如果  $MbaffFrameFlag$  为 1（当前宏块帧场自适应  $mb\_adaptive\_frame\_field\_flag$  为 1，并且当前图像片为帧编码，即  $field\_pic\_flag$  为 0）并且当前宏块是场宏块，则重建图像中样点亮度值为：

$$S'_L[xP + xO + j, yP + 2 \times (yO + i)] = U_{ij} \quad i, j = 0, \dots, 3 \quad (8.93)$$

② 否则（ $MbaffFrameFlag$  为 0 或者当前宏块是帧宏块），则重建图像中样点亮度值为：

$$S'_L[xP + xO + j, yP + yO + i] = U_{ij} \quad i, j = 0, \dots, 3 \quad (8.94)$$

如果  $U$  是色度块，对  $4 \times 4$  色度块中的每个样本元素  $U_{ij}$ ，根据变量  $MbaffFrameFlag$  进行以下操作：

① 如果  $MbaffFrameFlag$  为 1 并且当前宏块是场宏块，则重建中图像样点色度值为：

$$S'_C[(xP \gg 1) + xO + j, ((yP + 1) \gg 1) + 2 \times (yO + i)] = U_{ij} \quad i, j = 0, \dots, 3 \quad (8.95)$$

② 否则（ $MbaffFrameFlag$  为 0 或者当前宏块是帧宏块），则重建中图像样点色度值为：

$$S'_C[(xP \gg 1) + xO + j, ((yP + 1) \gg 1) + yO + i] = U_{ij} \quad i, j = 0, \dots, 3 \quad (8.96)$$

## 8.11 SP 片中的 P 宏块和 SI 片中的 SI 宏块的解码过程

第六章已经着重描述了 SP/SI 帧的应用、编解码基本原理和实验分析。本节结合草案进行 SP/SI 帧中的解码过程进行论述。鉴于片是 H.264 标准的基本编解码单元，因此实质上是阐述对 SP 片中的 P 宏块和 SI 片中的 SI 宏块的解码。

SP 帧分为主 SP 帧和辅 SP 帧，前者参考帧和当前编码帧属于同一码流，而后者则不属于同一码流。片头语义变量  $sp\_for\_switch\_flag$  用来区分主辅 SP 帧，0 表示主 SP 帧，1 表示辅 SP 帧。下面分别描述对这两种 SP 帧的解码过程。

### 8.11.1 主 SP 片中 P 宏块的解码过程

主 SP 片中 P 宏块的解码原理图如图 8.46 所示，具体又分为亮度变换系数和色度变换系数的解码过程。输入参数是当前宏块的帧间预测值和残差变换系数的预测值；输出参数为当前宏块的解码样点值。

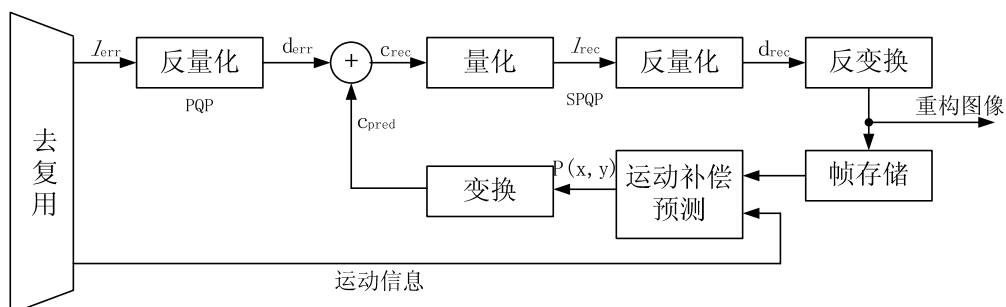


图 8.46 主 SP 帧解码原理图

### 8.11.1.1 亮度变换系数解码过程

输入参数：当前宏块的帧间亮度预测值  $\text{pred}_L$ 、残差变换系数预测值  $\text{LumaLevel}$  和变换系数  $\text{luma4x4BlkIdx}$

输出参数：当前宏块的解码亮度样点值  $u_{ij}$

亮度变换系数解码步骤：

- 1)  $\text{luma4x4BlkIdx}$  推导坐标  $(x, y)$ ，详见 Draft6.4.3
- 2) 求出宏块各点亮度的预测值：  $p_{ij} = \text{pred}_L[x+j, y+i]$   $i, j = 0 \dots 3$
- 3) 求  $p_{ij}$  的 DCT 变换

$$c^p = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \\ p_{30} & p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

(8.97)

- 4) 残差变换系数  $\text{LumaLevel}[\text{luma4x4BlkIdx}]$  逆扫描过程映射为二维矩阵  $c^r$

- 5)  $c_r$  反量化后与  $c_{ij}^p$  相加求得  $c_{ij}^s$

$$c_{ij}^s = c_{ij}^p + (((c_{ij}^r * \text{LevelScale}(\text{QP}_Y \% 6, i, j) * A_{ij}) << ((\text{QP}_Y / 6))) >> 6)$$

(8.98)

- 6)  $c_{ij}^s$  进行量化处理，但量化参数与  $c_{ij}^r$  不同

$$c_{ij} = (\text{Sign}(c_{ij}^s) * (\text{Abs}(c_{ij}^s) * \text{LevelScale2}(\text{QS}_Y \% 6, i, j) + (1 << ((14 + \text{QS}_Y / 6)))) >> (15 + \text{QS}_Y / 6))$$

(8.99)

- 7)  $c_{ij}$  反变换求得  $r_{ij}$

- 8)  $u_{ij} = \text{Clip1}(r_{ij})$

### 8.11.1.2 色度的变换系数的解码过程

输入参数：当前宏块的帧间预测色度值  $\text{pred}_c$  和预测残差变换系数  $\text{ChromaDCLevel}$  and  $\text{ChromaACLevel}$ ；输出参数：当前宏块的解码的色度值。

可以看出，色度得变换系数分 AC 和 DC 两种，但两种的区别只是 AC 系数采用了 DCT 变换，而 DC 系数采用了 Hadamard 变换，其余步骤基本一致，因此，下面只对 AC 系数的解码过程过行描述。

色度 AC 变换系数的解码过程和亮度类似，步骤如下：

- 1)  $\text{chroma4x4BlkIdx}$  推导坐标  $(x, y)$
- 2) 求出宏块各点色度的预测值  $p_{ij} = \text{pred}_c[x+j, y+i]$   $i, j = 0 \dots 3$
- 3) 求  $p_{ij}$  的 DCT 变换，得  $c^p$
- 4) 求变量序列  $\text{chromaList}[k]$

$$\text{chromaList}[k] = \begin{cases} 0 & k = 0 \\ \text{ChromaACLevel}[\text{iCbCr}][\text{chroma4x4BlkIdx}][k-1] & k = 1 \dots 15 \end{cases}$$

- 5) 变换系数  $\text{chromaList}[\text{chroma4x4BlkIdx}]$  逆扫描求得二维矩阵  $c^r$
- 6)  $c^r$  进行反量化, 量化参数为  $QP_C$  和预测值  $c_{ij}^p$ , 得  $c_{ij}^s$
- 7)  $c_{ij}^s$  量化处理, 但量化参数与  $c_{ij}^r$  不同, 量化参数为  $QS_C$
- 8)  $c_{ij}$  进行反变换得到  $r_{ij}$  样点值
- 9)  $u_{ij} = \text{Clip1}(r_{ij}) \quad i, j = 0 \dots 3$

### 8.11.2 辅 SP/SI 片的解码过程

辅 SP/SI 帧的解码原理如图 8.47 所示。同样又分为亮度变换系数和色度变换系数的解码过程。输入参数是预测残差变换系数值和预测样点值  $\text{pred}_L, \text{pred}_{Cb}, \text{pred}_{Cr}$ ; 输出参数是当前宏块的解码样点值。

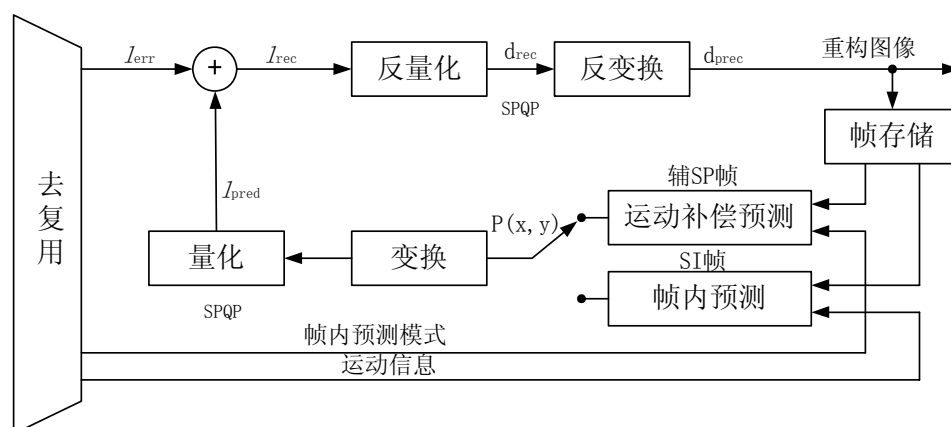


图 8.47 辅 SP/SI 帧的解码原理图

#### 8.11.2.1 亮度变换系数解码过程

输入参数：预测亮度样点值  $\text{pred}_L$  和亮度预测残差变换系数值  $\text{LumaLevel}$ 。

输出参数：当前宏块的解码亮度样点值

亮度变换系数解码步骤：

- 1)  $\text{luma4x4BlkIdx}$  推导坐标  $(x, y) (x, y)$
- 2) 求出宏块各点亮度的预测值:  $p_{ij} = \text{pred}_L[x+j, y+i] \quad i, j = 0 \dots 3$
- 3) 求  $p_{ij}$  的 DCT 变换
- 4)  $c_{ij}^p$  求进行量化, 量化参数为  $QS_Y$
- 5) 变换系数  $\text{LumaLevel}[\text{luma4x4BlkIdx}]$  逆扫描过程映射为二维矩阵  $c^r$
- 6) 求累加值:  $c_{ij} = c_{ij}^r + c_{ij}^s \quad i, j = 0 \dots 3$
- 7)  $c_{ij}$  反变换求得  $r_{ij}$
- 8)  $u_{ij} = \text{Clip1}(r_{ij}) \quad i, j = 0 \dots 3$
- 9)

### 8.11.2.2 色度的变换系数的解码过程

输入参数是预测色度值和预测残差变换系数值 ChromaDCLevel and ChromaACLevel, 输出参数是当前宏块的解码色度值

和上面类似, 色度得变换系数分 AC 和 DC 两种, 但两种的区别只是 AC 系数采用了 DCT 变换, 而 DC 系数采用了 Hadamard 变换, 其余步骤基本一致, 因此, 下面只对 AC 系数的解码过程过行描述。

AC 系数解码步骤:

- 1) chroma4x4BlkIdx 推导坐标(x, y)
- 2) 求出宏块各点色度的预测值  $p_{ij}$
- 3) 求  $p_{ij}$  的 DCT 变换, 的  $c^p$
- 4)  $c^p$  进行量化: 量化参数为  $QSc$
- 5) 求变量序列 chromaList[k]

$$\text{chromaList}[k] = \begin{cases} 0 & k = 0 \\ \text{ChromaACLevel}[\text{iCbCr}][\text{chroma4x4BlkIdx}][k - 1] & k = 1 \dots 15 \end{cases}$$

(8.100)

- 6) 变换系数 chromaList [chroma4x4BlkIdx ] 逆扫描求得二维矩阵  $c^r$
- 7) 求重构值:  $c_{ij}(\text{chroma4x4BlkIdx}) = c_{ij}^r(\text{chroma4x4BlkIdx}) + c_{ij}^s$
- 8)  $c^r$  进行反量化, 量化参数为  $QP_C$
- 9)  $c_{ij}$  进行反变换得到  $rij$
- 10)  $uij = \text{Clip1}(rij) \quad i,j = 0..3$

## 参考文献

1. “Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC,” in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050, 2003.
2. T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, “Overview of the H.264/AVC Video Coding Standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 560–576, July 2003.
3. S. Wenger, “H.264/AVC over IP,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 645–656, July 2003.
4. T. Stockhammer, M. M. Hannuksela, and T. Wiegand, “H.264/AVC in wireless environments,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 657–673, July 2003.
5. T. Wedi, “Motion compensation in H.264/AVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 577–586, July 2003.
6. M. Flierl and B. Girod, “Generalized B pictures and the draft JVT/H.264 video compression standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 587–597, July 2003.
7. T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, “Rate-constrained coder control and comparison of video coding standards,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 688–703, July 2003.
8. Schafer Ralf, Wiegand Thomas, Schwarz Heiko. “The emerging H.264/AVC Standard EBU Technical Review”, Jan. 2003
9. M. Karczewicz and R. Kurçeren, “The SP and SI frames design for H.264/AVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 637–644, July 2003.
10. H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, “Low-Complexity transform and quantization in H.264/AVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 598–603, July 2003.
11. P. List, A. Joch, J. Lainema, G. Bjøntegaard, and M. Karczewicz, “Adaptive deblocking filter,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 614–619, July 2003.
12. J. Ribas-Corbera, P. A. Chou, and S. Regunathan, “A generalized hypothetical reference decoder for H.264/AVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 674–687, July 2003.
13. Mathias Wien, “Variable Block-Size Transforms for H.264/AVC” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 604–613, July 2003.
14. Michael Horowitz, Anthony Joch, Faouzi Kossentini, “H.264/AVC Baseline Profile Decoder Complexity Analysis” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 704–716, July 2003.
15. Marta Karczewicz The. SP- and SI- Frames Design for H.264/AVC. *IEEE Trans. Circuits and Systems for Video Technology*, Vol 13, No.7, July 2003

## 第 9 章 H.264 视频编码传输的 QoS

### 9.1 互联网视频传输 QoS

Internet 在过去几年所取得的巨大成就和未来所蕴涵的巨大发展潜力几乎没有人怀疑。当人们在思考未来 Internet 的发展时，如何在 IP 网络上保证用户信息传输的质量就成为一个不容忽视的重要问题。目前，随着宽带网络的发展和用户需求的驱动，多媒体技术和相关的应用得到越来越多的关注，被认为是未来高速网络的主流应用之一。多媒体应用相对于传统 Internet 应用如 WWW、EMAIL 等，其对实时性更严，对带宽的需求更大。基于 IP 的多媒体应用分为三类：交互应用，如可视电话和视频会议；预编码的视频流下载；基于 IP 的视频流。为满足这些应用的需求，**必须解决四个 QoS 问题：吞吐量；传输时延；时延抖动；误码率**。同时，由于视频应用中大多采用的高压缩率的编码技术，其对传输误码的要求尤为苛刻，但是因为 Internet 的本质是尽力而为的网络，不提供传输的 QoS 保证，因此在视频应用中提高对传输误码的抗干扰和恢复能力是很有必要的，同时这也一直是多媒体通信领域研究的热点。

本节首先对错误恢复能力在 IP 视频通信中的重要性和实现途径进行简单地描述；然后对视频通信中传统的抗干扰和恢复方法[1]进行一个回顾，并为表述的更清晰，简单的介绍了通用的基于块的混合视频编码框架。

#### 9.1.1 错误恢复在视频通信中的重要性和实现途径

视频通信系统通常由五部分组成，视频源编码、复用/封装/信道编码、信道传输、解复用/解包/信道解码和视频解码。视频首先在编码器中进行压缩以减小码率，压缩的比特流然后被分割成固定的或变长的包，并与数据、音频等复合，然后打包，最后需要经过一个信道编码阶段，通常使用前向纠错编码(FEC)，用来防止传输的错误。在接收端，收到的包按照 FEC 解码并解包，输出的比特流被放入视频解码器用于重构原始的视频。在实际的使用中，许多应用在源端编码器中内嵌封装和信道编码作为一个网络适配层。数据包可能会由于流量拥塞或物理信道的损伤导致的比特错误而产生丢失或损坏，例如在当前的 Internet 和无线网络中；因此，设计一个能使压缩的比特流在传输发生错误时可以进行错误恢复的视频编解码方案很必要；同时也必须在编解码器与网络之间设计一个合适的接口机制以使编解码器能根据网络情况调整操作。

在视频通信中的错误控制由于下面几个原因而显得备受重视：

- 压缩的视频流由于在源编码器中采用了预测编码和可变长编码(VLC)，对传输的错误特别敏感。由于使用了空间和时间预测，一个带有错误的采样点将会导致同一帧或后续帧相关采样点的错误，同样由于VLC的使用，一个比特错误会导致解码器失去同步，从而使后续比特的正确接收失去意义。
- 视频源和网络环境通常是时变的，因此基于某些源和网络的统计模型设计出一个“优化”的解决方案是很困难的，甚至是不可能的。
- 视频源有着很高的码率，因此编解码器不可能过于复杂，尤其是对于某些实时应用。

**为使压缩的比特流对传输错误具有抗干扰性，必须在流中加入冗余数据，利用这些冗余数据和有效数据之间的关系，才有可能检测并修正错误。这个冗余可以加入到源端编码或信道编码中，通常采用源和信道联合编码实现，它在源编码和信道编码中统一分配一个一定总量的冗余。所有的错误恢复编码（error resilient encoding）方法基本上都符合这个前提，这些通常通过仔细的设计预测编**



码循环和变长编码器完成，用以限制错误扩散的范围。

即使一个图像采样或一块采样在传输中由于错误而丢失，解码器仍能基于周围的已经收到的采样点，利用空间和时间相邻的采样点的内在相关性，进行估计，从而达到错误恢复，这个技术即所谓“**错误隐藏**”技术。为保证解码器的错误隐藏，相邻采样点或块的压缩数据在封装时必须按照 interleaved 交织方式，用以提高损坏区域被未损坏区域包围的概率。**错误隐藏相对于错误恢复源端编码，优点是不需要增加额外的比特率，仅仅增加了解码器上的运算复杂度。**

最后，为保证源端编码器中的内嵌冗余的有效性和解码器的错误隐藏能力，**编解码器和网络传输协议必须相互协作**。例如，如果比特流中的某些比特比其他的更重要，重要的部分在网络分发时必须被赋予更高的 QoS 参数集。为抑制错误扩散，网络必须提供反馈信道，这样编码器就能知道在解码器中那部分重构信号损坏，在预测将来采样点时就不使用这些信号。

总体来说，**抗传输误码的设计机制分为三类**：在源端编码和信道编码中引入**冗余数据**的方法，使比特流对可能的错误具有更强的恢复能力；在解码器中基于错误检测来对**错误的影响进行隐藏**；通过**源端编码器和解码器之间的交互**，来使编码器根据解码器检测到的丢失情况调整操作。在本文中，我们将它们统称为错误恢复技术 (ER techniques)。

下面将要简单回顾一下基于块的时间预测和变换编码的编码器。因为这些编码器是迄今最实用和有效的编码器，并已被国际视频编码标准采纳。

### 9.1.2 基于块的混合视频编码框架

图 9.1 显示了编码框架中的关键步骤。如图所示，每个视频帧被分成固定大小的块，每个块进行独立的处理，因此命名为“基于块的”。“混合”意味着每个块使用运动补偿时间预测和变换编码的联合编码方法。也就是说，一个块首先通过前一个编码参考帧中的匹配块预测，最佳匹配块位置的估计即“运动估计”，当前块和匹配块之间的位移即“运动矢量”(MV)。基于 MV 预测一个块的过程称为“运动补偿”。块的预测误差（残差），通过 DCT 变换后，对相应的系数进行量化，并使用 VLC 将它们转换成二进制码字，使用 DCT 的目的在于减少相邻像素点误差的空间相关性。因为大量的高频系数在量化后为 0，VLC 采用游程编码的方法，使用 zig-zag 扫描将系数排序进一个一维数组，这样低频系数放置在高频系数之前。这种方法中，量化后的变换系数通过非零值和其前面的零的数目来表示。每个不同的符号都代表着一对零游程和非零值，它们使用变长码字进行编码。上述的讨论都假设时间预测是成功的，即预测残差块只需要比原始图像较少的比特。这种编码方式称为 P 模式。当情况不同时，原始的块将直接采用 DCT 和游程编码进行编码，这种编码方式称为帧内模式 (I 模式)。除了使用一个参考帧来进行预测的方法外，也可以使用双向预测的方法，即寻找两个最佳的匹配块，其中一个位于前向序列中，一个位于后向序列中，并使用两个匹配块的权重均值作为当前块的预测值。这种情况中，两个运动矢量与每个块相关，这种编码方式称为 B 模式。P 模式和 B 模式又统称为帧间模式。模式信息、运动矢量和其他枝节信息（例如图像格式、块位置等）也通过 VLC 编码。

在实际使用中，用于运动估计的块大小不一定与用于变换编码的相同。通常运动估计一般采用较大的块，即宏块 (MB)，它可以再划分为一些块。例如在大部分的视频编码标准中，宏块的大小为 16x16 点阵，而块大小为 8x8 点阵。编码模式在宏块级进行决定。因为相邻宏块的运动矢量往往类似，当前宏块的运动矢量首先使用前一个宏块的运动矢量来进行预测，同样块的 DC 系数采用同样的方法处理。在所有的视频编码标准中，一些宏块构成宏块组 (GOB) 或片，一些片构成一帧。片的大小和形状在不同的视频编码标准和图像大小中也不同，并可以根据应用需求进行调整。**运动矢量和 DC 系数的预测通常局限在同一片中**。如果一帧都采用帧内模式编码，这一帧就称为 I 帧。一般用于一个序列的第一帧。在使用高码率或对实时性限制较松的应用中，会周期地使用 I 帧来阻止潜在的错误扩散，并提供随机的存取。低时延应用不能采用这个有效方法，因为 I 帧通常比任何预测帧大好几倍。P 帧只使用过去的帧进行预测，一个 MB 既可以通过帧内模式也可以通过 P 模式编

码。最后，一个 B 帧使用双向预测，一个 B 帧内的 MB 可以使用 I 模式、P 模式或 B 模式编码。一个 B 帧只能在周围的 I 帧或 P 帧编码后再进行编码。

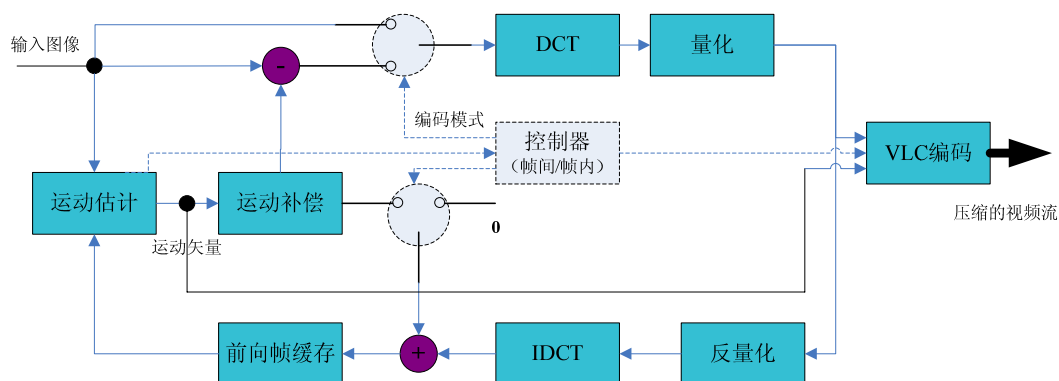


图 9.1 基于块的混合视频编码框架图

从图 9.1 的基于块的混合视频编码框架可以看到，由于采用了帧内和帧间预测的模式，尤其是 P 模式（单向帧间预测）和 B 模式（双向帧间预测）的大量使用，视频流压缩效率大大提高，但随之而来的是，由于传输误码甚至丢包所导致的影响会迅速扩展，这使采用误码恢复技术更具重要性。

### 9.1.3 视频通信中提高 QoS 的抗误码和错误恢复技术

在错误恢复编码中，编码器通常在编码流中加入适量的冗余比特，因此，ER 编码器效率不是最高的。其设计的目标在于使用最小的冗余来获得错误恢复的最大增益。有很多方法用于添加冗余比特，一些有益于防止错误扩散，另一些用于帮助解码器通过错误监测进行更好的错误隐藏，还有一类目标在于保证基本级别的质量，并提供传输错误发生时质量比较平滑的下降。通常错误恢复编码技术分为鲁棒熵编码和错误恢复预测两类。

鲁棒熵编码是一种常用方法。由于使用 VLC 编码，因此一个码字中间任意一个比特错误或丢失都会导致这个码字以及后续码字不能解码。一个简单有效的提高码流鲁棒性的方法就是在比特流中周期的插入重同步标记。这样，解码器可以在检测到重同步标记之后继续正确的解码。然而很明显，重同步标记的插入会降低编码效率，首先，标记越长，使用频率越高，占用的比特就越多；其次，标记的使用通常会中断图像内预测机制，例如 MV 或 DC 系数的预测，同样会增加更多的比特。但是，标记越长，使用频率越高，解码器就能更快的获得重同步，这样一个传输错误就会只影响一个更小的区域。因此，在实际的视频编码系统中，一般使用相对较长的同步码字。

#### 9.1.3.1 反向变长编码

反向变长编码 (RVLC)[3,4]是另一种常用方法，其原理图参见图 9.2。上面我们都假设一旦错误发生，解码器将丢掉所有比特直到获得重同步。但是如果使用 RVLC，解码器就能在重同步前通过如图 2 所示的反向操作进行解码。这样，更少的正确接收的比特被丢弃，传输错误影响的区域将会变小。通过提供前向解码器和后向解码器之间的交叉检查的能力，和适度的复杂度增加，RVLC 能帮助解码器检测到使用非逆向 VLC 不能检测到的错误，并提供错误位置的更多信息，因此减少了不必丢弃的数据量。RVLC 和同步标记插入已经被 MPEG-4 和 H.263 采纳。

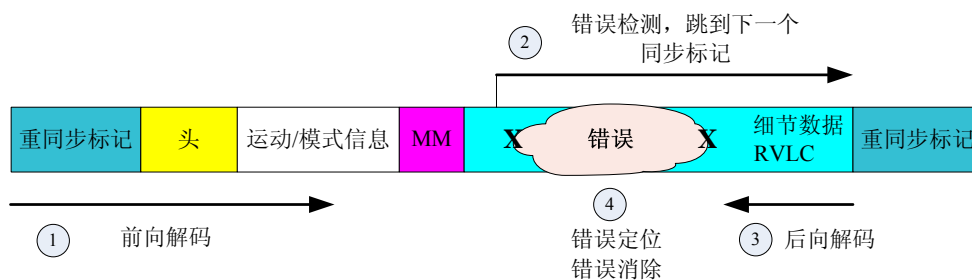


图 9.2 反向变长编码解码原理图

错误恢复预测技术是另外一种常用方法。我们知道压缩视频对传输错误敏感的一个主要原因就是使用了时间预测和空间预测。一旦有误码发生，在解码器上重构的帧将会与原图像有较大差别，这样会导致使用这一帧作为参考帧的后续帧产生错误。DC 系数和 MV 使用了空间预测，这同样会导致错误扩散，尽管这种扩散限制在一帧内。在绝大多数的视频编码标准中，这样的空间预测都进一步的限制在一帧中的子区域（块组或片）中。在实际的编码系统中通常采用下面四种方法解决上述问题：插入 I 模式的宏块或帧；独立的段预测；带有非均衡错误保护的分层编码；多描述编码（multiple description coding，MDC）。

阻止时域错误扩散的一个方法是周期的插入帧内编码的帧或宏块[6]。对于实时应用由于时延限制通常不可能采用一帧的帧内编码。然而使用足够数量的帧内编码的宏块却是一个有效的具有高扩展性的错误恢复工具。当使用宏块的帧内编码作为错误恢复的目的时，宏块的数量和它们的空间位置需要仔细决定。必需的帧内编码的宏块的数量很明显由连接的质量决定。大量的实用系统提供关于网络质量的带外信息，或 heuristic 的方法用于获得这类信息，例如无线环境中的天线信号强度和因特网连接中的 RTCP 接收器报文。对于 I 模式块的空间位置有多个方案，位置随机分配是一种有效方法，同样位置分配于高活动区域（由 MV 的平均幅度值决定）同样有效。目前所知最好的方法是使用基于丢包率的率失真优化方案[7]来决定 I 模式宏块的数量和位置。最后还有一个闭环的方法，使用一个反馈信道将丢失或损坏的宏块数据信息传达给发送端，用于触发帧内编码。

非均衡错误保护的分层编码属于错误恢复预测技术的一种。另一个限制错误扩散范围的方法就是将数据分成多个段，仅在同一段内部进行时间和空间预测。一个途径是将奇数帧作为一段，偶数帧作为另一段，这种方法在[8,9]中称为视频冗余编码。它同样可作为完成下面即将阐述的多描述编码的一个方法。另一个途径是将帧分为多个区域（GOB 或片），某个区域只能由前一帧的相应区域进行预测，这在 H.263 中称为独立段解码(ISD)。

### 9.1.3.2 分层编码

分层编码(分级编码，LC)是指将视频编码成一个基本层和一个或多个加强层。基本层提供了一个较低的但可接受的质量，每个附加的加强层逐步的提高质量。本质上，LC 是一个向用户针对同一视频提供不同的质量等级，占用不同带宽和不同的解码能量的方法。因此 LC 也称作 scalable 编码。用于错误恢复时，LC 必须与传输系统中的非均衡错误保护(UEP)相结合，因此基本层使用更强的保护，例如分配更可靠的子信道，使用更强的 FEC 编码，或允许更多的重传等[10,11]。UEP 的实现有网络层的和应用层的，目前绝大部分网络不提供支持，相对来说应用层的实现比较灵活，如使用基于包的 FEC 等，只是需要另外的比特。

在使用 LC 的编码方法中，可以通过编码级别，来提高视频的解析度、帧频和图像的质量。视频的可分级性就是指能够同时获得多于一种解析度或图像质量的能力。由于用于多个编码流，那个基本比特流称为基本层（B 级），其它的比特流均称为加强层（E1 级，E2 级等）。基本层中包含了解码必须的重要信息，如头信息，运动矢量和 DC 分量等。因此加强层必须在基本层正确的前提下才能完成解码操作。由于加强层以基本层为参考的主要或唯一来源，因此由加强层的数据报丢失而导致

的错误扩散将被最小化。在 H.263 和 MPEG-4 有三类分级方式：时间、信噪比（SNR）和空间。

- SNR 分级：在 SNR 分级情况中，基本层码流由于使用了更高的压缩，获得的视频质量较低，而加强层码流按照较高的质量进行编码。其它参数，如空间和时间的解析度，基本层与加强层相同。
- 空间分级：在空间分级情况中，加强层用于提高图像的解析度。例如，基本层的解析度为 QCIF，而加强层则为 CIF 解析度。这样，当输出需要 QCIF 解析度时，就只需要基本层的码流被解码；而当输出需要 CIF 解析度时，基本层和加强层都必须被解码。但是它的编码复杂度比其它的编码方法复杂。
- 时间分级：在基于对象的时间分级中，某个给定的对象的帧频被提高，这样它就能拥有比其它区域更平滑的移动。也就是说，给定的对象的帧频比其它区域的帧频。这个编码方法的复杂度是分级编码中最小的，因此这个分级方法在 Windows Media Player 使用。

有两类加强层编码方法，一类称为 PPP，仅由 P 帧组成，如图 9.3；另一类称为 PBB，由 P 帧和 B 帧组成，如图 9.4。在 PPP 方式的加强层编码过程中，它仅参考基本层中的一个帧用于运动估计，这样加强层以基本层或同层的一个帧的上采样作为参考，它减少了估计时间，但码率较高，使用于 FGS 中。而在 PBB 模式中，同时参考基本层和加强层，有着更高的压缩率，但增加了帧之间的复杂度，同样，这种方法可以有效地阻止错误扩散。

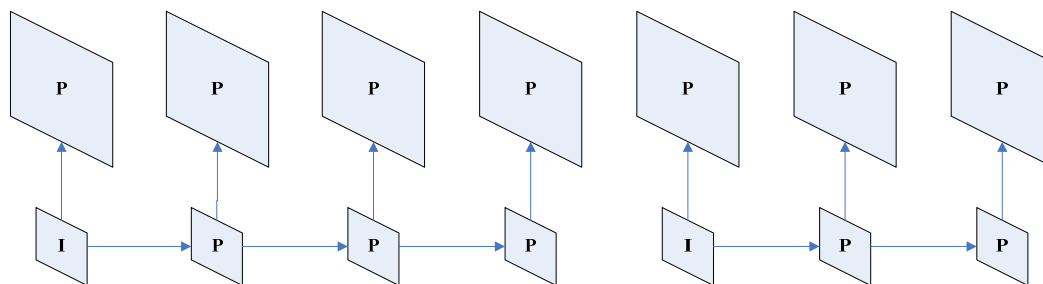


图 9.3 只含 P 帧的增强层格式

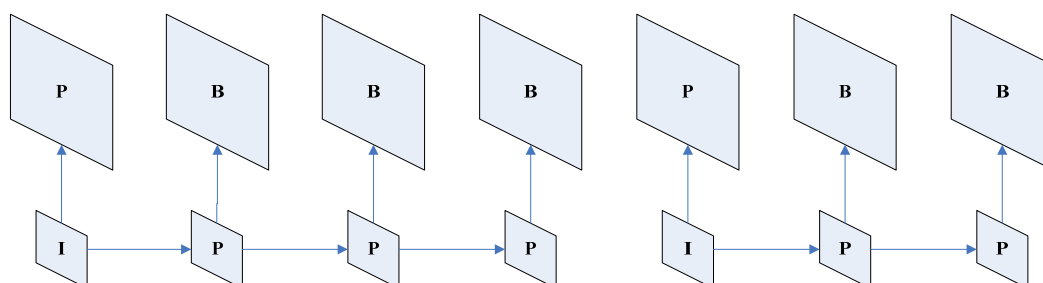


图 9.4 含 P 帧和 B 帧的增强层格式

### 9.1.3.3 多描述编码

同 LC 一样，MDC 也将一个源编码成多个子流，称为描述，区别是产生的描述是相关的，有着同样的重要性。任何一个单独的描述必须提供一个基本级别的质量，多个描述一起提供改善的质量。因此所有的描述必须共享源的一些基本信息。这种相关性使解码器能够根据正确接收的一个描述估计一个丢失的描述，因此能从任意一个描述提供一个可接受的视频质量。另一方面，这种相关性就是 MDC 中源端的冗余。MDC 相对于 LC 的一个优点是它不需要网络提供可靠的子信道。

多描述视频编码是一个可用于解决尽力而为的网络所引起的 detrimental 影响的方法。在一个多描述系统中，视频序列被编码成两个或更多个互补码流，且每个码流独立解码。当所有互补码流相结合时，视频流提供最高的质量级别，但即使分别解码，它们也能提供可接受的质量级别。这些流可通过各自的网络路径传送，从而经历或多或少的各自的丢包率和延迟。当一个流的某一部分丢失



或延迟太大时，视频的播放将不会受到严重地影响而引起完全停止等待重新缓冲。相反，其它的流可以继续播放，仅仅总体的视频质量有轻微下降。

在两个独立流的情况下，如图 9.5，问题可以描述如下：一个给定的源被编码成两个流，码率分别是  $R_1$  和  $R_2$ ；这两个流分别通过两个网络信道进行传送；接受端由三个独立的子解码器组成，当两个流全部按时到达时，中间的解码器输出图像序列，当其中一个流按时到达时，两个边缘解码器其中之一输出图像序列。中间解码器输出的失真定义为  $D_0$ ，两个边缘解码器的输出失真分别定义为  $D_1$  和  $D_2$ 。状态选择器从最适当的解码器输出视频流。



图 9.5 两个独立流的 MD 系统模型

目标在于对于给定情形下寻找最优的性能， $\{R_1, R_2, D_0, D_1, D_2\}$ 。这五个因素的每个的相对重要性根据当时的情形而定。在某些系统中，可靠性最重要，此时  $D_1$  和  $D_2$  变得更重要，比特率的重要性可能会下降；在另外的情况中，可能正好相反，编码效率的鲁棒性显得更迫切。系统不必对任何一个流平衡，即  $R_1$  可以拥有比  $R_2$  更大的权重等。

作为 MD（多描述）系统的一个简单例子，一个序列被分割成两部分，所有的奇数场作为一个流，所有的偶数场作为另一个流，这个过程产生了两个有着一半垂直解析度的新序列。然后这两个序列可以被独立编码和进行网络传输。当两个流均按时接收，两个流的行通过交织从而重构一个带有失真的完整解析度序列。如果一个流丢失，滤镜（deinterlacing）算法可用于估计丢失的行。通常，估计的失真（ $D_1$  或  $D_2$ ）将比两个流同时接收所得到的大，但尽管一个流全部失效（outage），播放仍能继续。

当然，鲁棒性的获得需要付出代价。对帧进行空间下采样会降低空间的相关性。根据上面的问题所对应的公式，中间解码器要获得某个失真  $D_0$  所需的总体码率  $R_1+R_2$  肯定大于某个单独的流编码器达到  $D_0$  所需的码率。然而，根据特定的应用考虑，放弃一些效率从而获得可靠性的增加，是很有必要的。

这儿需要注意的是多描述编码与分级视频编码的区别。与 MD 编码一样，一个分级编码器将一个序列编成多个流，称为层；然而分级编码中，一个独立的基本层后面是一个或多个不独立的增强层，（参见图 9.6）。这就允许某些接收者只对基本层解码而获得基本视频，同时另一些接收者同时对基本层和一个或多个加强层同时解码，从而获得改善的图像质量、空间解析度或帧频。与 MD 编码不同，基本层的丢失会导致加强层无用。

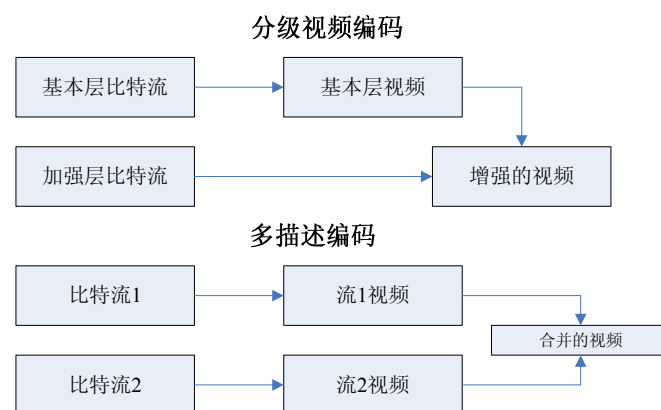


图 9.6 LC 系统模型和 MD 系统模型之比较

#### 9.1.3.4 解码器错误掩盖

解码器错误掩盖是指对传输错误导致的丢失信息的恢复和估计。在基于块的混合编码范例中，在损坏的 MB 有三类信息需要估计：纹理信息，包括原始图像块和预测错误块的像素或 DCT 系数的值；运动信息，包括 P 模式或 B 模式编码的 MB 中的 MV；宏块的编码模式。众所周知，原始图像的能量主要集中在低频分量，例如空间和时间相邻的像素点的色度值的变化很平滑，除了在区域的边缘。所有的用于恢复纹理信息的技术都是基于上述特性进行某种空间、时间内插。MV 域也采取同样的方法，对于编码模式，开发的方法倾向于 heuristics。

对于纹理信息的恢复，有四种方法：运动补偿时间预测；空间内插[19]；基于重构图像最平滑的空间和时间内插[21]；使用 Projection onto Convex Sets (POCS) 技术的空间内插[22,23]。运动补偿时间预测是一个简单有效的恢复损坏 MB 的方法，即基于宏块的 MV 复制前一帧中相应的宏块，这种方法需要 MV 是有效的；当 MV 也丢失时，先要估计 MV，其方法下面会描述；为减少估计 MV 中误差的影响，时间预测将与空间内插相结合。空间内插是根据相邻的正确接收的块中的像素点内插估计损坏的块中的像素点，通常只使用相邻块中的边缘像素点用于内插估计；一个更简单的方法是估计损坏块的 DC 系数（例如均值方法）[20]，并使用一个等于估计 DC 值的常量替换整块，DC 值的估计可以通过周围块的平均获得。在进行空间内插时存在一个问题就是如何决定合适的内插过滤器

(interpolation filter)，另一个问题是不能忽略正确接收的 DCT 系数，这些问题可以如下解决：使损坏块中的像素点平滑的连接到它在时间和空间上相邻的像素点，如果部分的 DCT 系数正确接收，估计必须保证恢复的块尽可能的平滑，并满足恢复的块的 DCT 与接收到的系数相同的条件，上述的目标可以公式化成非限制性优化问题，在不同丢包率的模型中解决方案对应到不同的时域、空间域和频域的内插过滤器。此外还有一个进行空间内插的方法是使用 POCS 方法，但是由于基于 POCS 估计的方法通过一个迭代的过程进行，因此对于实时应用不适用，本文不多加讨论。

编码模式和运动矢量同样可能损坏。一个估计损坏 MB 的编码模式的方法是收集相邻的 MB 的编码模式的统计信息，并选择一个最可能的模式 [24]。另一个简单保守的方式就是假设 MB 是采用帧内模式的，仅使用空间内插用于恢复下面的块[21]。对于估计丢失的 MV 有下面几个简单的方法[25]：(a) 假设丢失的 MV 为零，这对于相对变化较小的视频序列很有效；(b) 使用前一帧中的对应块的 MV；(c) 使用空间相邻的块的 MV 的均值；(d) 使用空间相邻的块的 MV 的中间值；(e) 重新估计 MV [19]。通常如果一个宏块损坏了，那么其水平相邻的 MB 也一般已经损坏，因此平均和取中间值主要针对垂直方向。实验显示最后两个方法能取得最好的重构结果[26]。

#### 9.1.3.5 编码器和解码器的交互错误控制

通过编码器和解码器的交互进行错误控制是一个有效方法。当在编码器和解码器之间建立一个反馈信道时，解码器可以通知编码器哪块数据损坏，这样编码器可相应调整操作来抑制甚至消除错误造成的影响。当网络的时延允许的情况下，在实时交互应用中甚至可以采取 ARQ 机制，重传损坏的参考帧，只要这些数据在解码器使用之前到达，就能部分的消除错误影响。用于编码器基于解码器反馈的信息调整操作的方法有基于反馈信息的参考帧选择和基于反馈信息的错误跟踪。

一种有效利用反馈信道的途径就是参考帧的选择 RPS[27]。如果编码器通过反馈了解某个编码帧 N 受到损坏，那么它可以决定下一个 P 帧不使用帧 N 作为参考帧，而使用更前一帧（已经被解码器正确接收）。这要求编码器和解码器存储多个过去的解码帧。关于使用的参考帧在比特流中传递。相对于使用 I 帧，这种方法效率更高，同时使用 RPS 并不意味着更多的时延。当然，随着反馈信息分发时间的变长，编码效率就会更多的丢失。

除了使用更早的帧作为参考帧外，编码器可以跟踪第N帧中损坏的区域对其后d帧的影响，基于反馈信息的错误跟踪，并采取下列某种方法：(a) 将帧n+d中将要使用帧n+d-1中的损坏像素点进行预测的区域使用I模式；(b) 避免使用帧n+d-1中的损坏区域来预测帧n+d；(c)对帧n+1 到n+d-1进行与解码器相同的错误隐藏，这样编码器中的参考帧在进行帧n+d编码时将与解码器一致。最后一个方法相对复杂，第一个方法最简单，但效率最低，实验显示它比RPS方法效率低[28]。

## 9.2 无线网视频传输 QoS

在过去二十年里，移动通信和多媒体通信获得了史无前例的发展，并获得了巨大的商业成功。移动通信和多媒体技术的融合正在加速进行，在诸如网络架构、低功耗的集成电路、功能强大的数字信号处理芯片、高效的压缩算法等方面的研究成果不断涌现。面向无线通信的视频图像编码与传输技术已成为当今信息科学与技术的前沿课题。

无线网络除了提供语音服务之外，还提供多媒体、高速数据和视频图像业务。无线通信环境(无线信道、移动终端等)以及移动多媒体应用业务的特点对视频图像的编码与传播技术提出了新的要求。首先，传输带宽一直是至关重要的，这也对无线视频通信网络的架构、通信协议以及视音频编码标准的研究产生了重要的影响。多媒体信息的信息量十分庞大，例如广播级质量的数字电视信号码率未压缩时为 216Mbps，即使采用 CIF 格式也需达 37Mbps。采用当前最新的视频编码标准，广播质量的视频图像需要的带宽大约为几 Mbps；而对于诸如可视电话这类低分辨率和图像相对静止的视频，要获得令人满意的图像质量至少需要几十 kps 的带宽。与此同时，当前广泛使用的全球移动通信系统 GSM，通常只能提供 10-15kbps 的带宽，刚够满足压缩语音编码的需求，对于视频传输是远远不够的。幸运的是，随着第三代移动通信技术 3G 以及未来移动通信技术的发展，视频传输的带宽瓶颈会得到进一步的缓解。

2003 年 ISO/IEC 的运动图像专家组 (MPEG) 与 ITU-T 的视频编码专家组 (VCEG) 在再次联手制定了最新的视频编码标准 H.264/AVC。其主要目的和特点是更高的编码效率和更好的网络适应性。在相同重构图像质量下，与 H.263+ 和 MPEG-4 ASP 标准相比，节约 50% 的码流；采用分层模式，定义了视频编码层 (VCL) 和网络提取层 (NAL)，后者专为网络传输设计，能适应于不同网络中的视频传输，进一步提高网络“亲和性”。引入了面向 IP 包的编码机制，有利于网络中的分组传输，支持网络中视频的流媒体传输。具有较强的抗误码特性，特别适应丢包率高、干扰严重的无线视频传输的要求。

### 9.2.1 无线视频通信系统

图 9.7 给出了无线视频通信系统的框图。发送端，视频信源的输入视频经视频编码器压缩到特定的码率的码流，经过传输层和网络层的封装打包后，进行信道编码，信道编码过程引入信道错误控制措施：前向纠错编码 FEC、自动请求重发 ARQ 和数据交织 Interleaving，增强码流在无线信道的传输性能，信道编码输出的经调制在无线信道上传输；接受端，进行解调和信道解码后，再进行解包处理，得出编码视频流，进行视频解码，输出的重建的视频信号，可根据应用进行视频图像后处理。

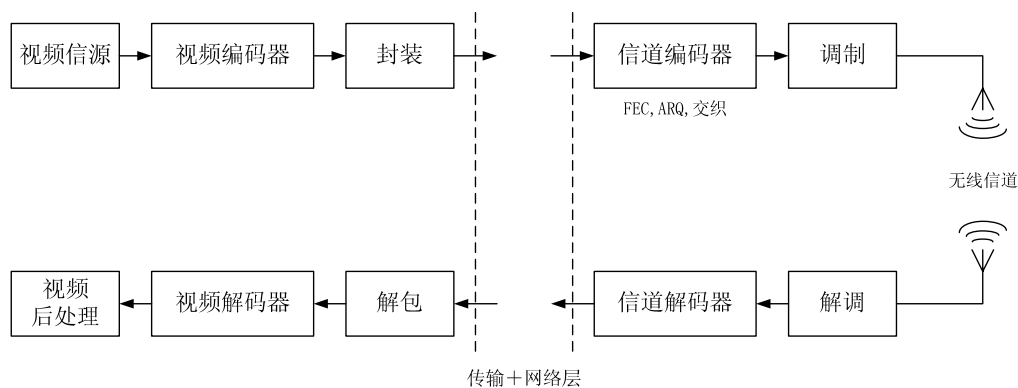


图 9.7 无线视频通信系统框图

从上图可以看出，无线视频通信系统与有线视频通信系统的区别在于，其包含一个无线的链路。众所周知，无线信道的有效传输带宽是非常宝贵的，而且各种噪声干扰也会影响无线信道的数据传输质量。因此，无线信道成为了无线视频传输关注的一个焦点。无线传输中面临的衰落和突发性错误必须采用多种错误控制技术处理，这其中典型的有前向纠错编码 FEC 和 ARQ。FEC 和数据交织结合明显消弱了无线信道突发性错误的影响，而基于反馈信道的 ARQ 则可正确的恢复错误的数据，避免错误的传播。这些错误控制技术作为信道编解码的组成部分，在下一节进行详细描述。

## 9.2.2 无线信道编码和错误控制

要实现无线环境下的视频通信，首先必须搞清无线信道的物理特性，因此下面，首先对无线信道的特性进行描述，同时为了便于理论研究和模拟仿真，建立了三种常用的无线信道模型。众所周知，信道的编解码策略对于无线信道传输的质量至关重要，在此分别讨论信道编解码的错误控制技术，其中主要包括前向纠错编码 FEC。数据交织和自动重发请求 ARQ。最后，讨论了 IP 协议在无线通信中的应用。

### 9.2.2.1 无线信道的特性

通信信道有有线信道和无线信道。有线信道可视为恒定参量的信道，是人为制造的信道，它从起初的明线、到同轴电缆，直到今天的光纤组成的光缆，通信容量越来越大，通信质量也越来越好，其性能指标正逐步趋向理想的传输信道。但无线信道却与之不同，是开放的变参量信道。无线通信的是以传输环境中的电磁波传播为基础，发送的电磁波有散射、折射和衰落。各种散射成分的干扰，形成分布不规则的场强，因而接收信号衰落且失真。这种干扰的严重程度取决于散射环境的具体物理性能。

因此，无线信道环境恶劣，实现无线网络上传高质量的视频是一个富有挑战性的任务。存在以下问题：

- 有限的带宽

当前的移动通信网络为用户提供的传输带宽仍是非常有限的，尽管这个问题在 3G 会得到某种程度上的缓解，考虑到无线传输信道的容量有限，多媒体信息的信息量又异常庞大，而且，目前市场上最终用户大部分是按照流量付费的方式来使用无线网络数据服务的，视频编码标准的高压缩效率就显得尤为重要。

- 带宽波动

无线信道的吞吐量会因为多径衰落、同频干扰和噪声扰动会造成无线信道的吞吐量降低，而且基站和移动终端之间距离的变化以及不同网络之间的移动等情况均会造成信道带宽发生剧烈波动，



严重影响无线网络上的实时视频传输。

- 高误码率

与有线信道相比，无线信道一般具有大得多的噪声，而且多径和阴影衰落，使得误码率（BER）非常高，误码严重影响视频传输的质量，因此，视频编码标准是否具有很强的抗误码能力成为了确保无线视频传输 QoS 的关键之一。

### 9.2.2.2 信道模型

要实现无线视频传输，首先需要了解无线信道的传输特性，建立一个既实用又有一定精度的无线信道模型是十分必要。下面总结了几种在理论研究和系统仿真中经常使用的几种无线信道模型。

- 高斯白噪声衰减模型(AWGN)

加性高斯白噪声 AWGN（Additive White Gaussian Noise）是最常见的一种噪声，它表现为信号围绕平均值的一种随机波动过程。加性高斯白噪声的均值为 0，方差表现为噪声功率的大小，一般情况下，噪声功率越大，信道的波动幅度就越大，接收端的信号的误码率也就随着越大。研究通信系统的误码率与信道质量的关系，加性高斯白噪声信道的研究是基础。下面假设为在整个信道带宽下功率谱密度为常数，且振幅符合高斯概率分布，则高斯信道的概率函数为：

$$P(v) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{v - m_v^2}{2\sigma^2}\right) \quad (9.1)$$

- Markov 模型

由于用户的移动和外界自然条件的变化，无线信道容易产生 Rayleigh 衰减和多用户干扰，传输码流往往会引起突发性错误，表现为几个到几十个毫秒时间内连续出现误码，误码持续时间与环境特性以及接收端的运动速度有关，在此原理基础上建立了 Markov 模型，图 9.8 给出了最简单的二阶的 Markov 无线信道模型。其中，信道有两个状态  $S_0$  和  $S_1$ 。当信道处于状态  $S_0$  时，报文正确传送和接收，当信道处于状态  $S_1$  时，报文将会出错。突发性错误造成的连续出错报文的数量由信道处  $S_1$  的时间长短和每个报文大小所决定。状态  $S_0$  和状态  $S_1$  之间的状态为

$$P = \begin{Bmatrix} 1 - P_{01} & P_{01} \\ P_{10} & 1 - P_{10} \end{Bmatrix} \quad (9.2)$$

其中， $P_{01}$  表示信道传输状态由  $S_0$  到  $S_1$  的概率， $P_{10}$  表示信道传输状态由  $S_1$  到  $S_0$  的概率。

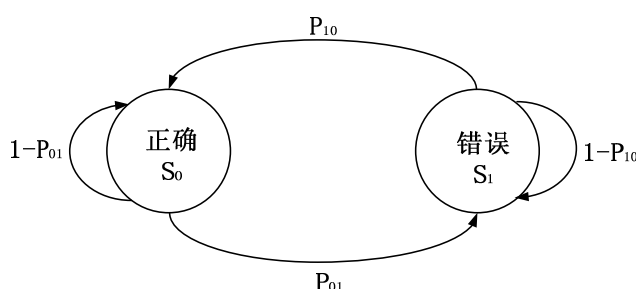


图 9.8 二阶 Markov 无线信道模型

- 随机出错模型

为了克服无线传输中的突发性出错，可以通过交织编码(interleaveing coding)，将突发性出错转换为随机性出错。要实现出错类型的转变，需要一定的交织编码深度，但交织编码深度越大，带来的延时越大，限制了交织编码在实时传输中的应用。

在基于 IP 网络中，随机性出错会带来报文丢失。位出错率和报文丢失率满足以下关系：

$$P_e = 1 - (1 - p)^l \quad (9.3)$$

其中  $p$  为位出错率， $l$  为报文长度。若采用了纠错容量为  $t$  的 FEC 编码，则报文丢失率可以表示为

$$P_{ec} = 1 - \sum_{i=0}^t \binom{l}{i} p^i (1 - p)^{l-i} \quad (9.4)$$

### 9.2.2.3 信道编码和错误控制

在这节，将主要讨论信道的编码和错误控制技术：FEC 和 ARQ，后者需要编解码器之间存在一条反馈信道，而前者则不需要，同时为了增强 FEC 处理突发性错误的效果，对数据交织的原理进行了阐述。

- 前向纠错编码（FEC，Forward Error Correction）

前向纠错技术是一种重要的纠错编码方式，在视频传输容错技术中得到了广泛的应用，常用的有 BCH 码和 RS 码。在 H.263 标准中使用了(511, 493)的 BCH 码，可以检测并纠正 2bit。但是 FEC 明显的缺点是降低了编码的效率。同时 FEC 的适用范围也是有限的，它只对平均分布的随机性位错比较有效，对于较高的误码率或者突发性错误效果较差。而无线传输系统中，突发性的错误是常见的，因此为了使 FEC 在无线视频传输中能发挥容错效力，**其实际应用中 FEC 编码、数据随机化和数据交织相互结合，共同作用。**

为了实现良好的容错性能和较高的编码效率，需要根据信道的传输特点如位出错率、报文出错率、传输速率、出错类型等对 FEC 编码策略进行调整。视频比特流中不同部分的重要性不同，报头(包括视频图像的尺寸、与解码有关的时间戳以及编码的格式)和 MV 信息比 DCT 系数对视频质量的影响要大，同样，低频的 DCT 系数要比高频的 DCT 系数对视频质量的影响大。对重要部分优先保护称为不等保护(UEP)。在 H.264 中，可以采用数据分割(Data Partition)的编码模式，将片的编码信息分为运动向量和 DCT 系数两部分，这样就可以优先对运动向量进行 FEC 编码，充分发挥 FEC 的纠错能力，提高编码效率。此外，FEC 还可以和自动重传(ARQ)或分层编码模式相结合，发挥更大作用

- 数据交织

许多容错算法能够很好地解决平均分布的随机性位错误，但对于突发性错误和某一段时间内的连续错误，效果就很差了。而**数据交织就是一种把码流中突发性错误转变成随机错误的方法。**

数据交织实质上就是一个置换器，目的就是要最大程度地置乱原数据排列顺序，避免置换前两个相邻数据在置换后再次相邻。

图 9.9 描述了数据交织的基本原理。其中，假定输入的码流  $P_0=\{1,2,4,\dots,15\}$ ，经过图 9.9(a)的交织器，读出的码流为  $P_1=\{1,6,11,2,7,\dots,15\}$ ，在解交织时，按图 6(b)的解交织器，正好可以输出码流  $P_2=P_0=\{1,2,4,\dots,15\}$ 。图 9.9(c)给出了在传输过程中发生了突发性错误的情况，可以看出，没有经过交织的比特流的错误是连续的，对于这种错误情况，诸如 FEC、错误隐藏等容错算法的效率很低，而经过交织处理的比特流中，突发性错误的已经分别开，这就有利于其他容错的算法的实现，同时，从图 9.9(c)中可以看出，交织的效果与交织器的设计  $M \times N$  紧密相关，最佳的交织器的设计能够实现相邻位置，交织后再次相邻。这样可以充分利用错误位周围正确接收位的相关性，从而纠正恢复错误位。

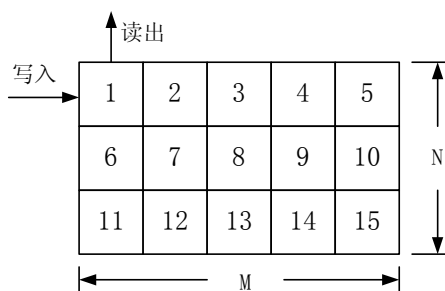


图 9.9(a) 交织

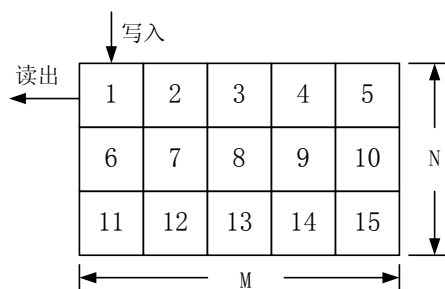


图 9.9(b) 解交织

无交织的比特流	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
有交织的比特流	1	6	11	2	7	12	3	8	13	4	9	14	5	10	15
无交织的接受比特流	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
去交织的接受比特流	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

图 9.9(c) 无交织和有交织处理突发错误的比较

## ● ARQ

自动请求重发 ARQ 是依靠反馈信息重传出错报文的差错控制方法。其实现过程如下，首先发送端发送数据报文，经信道传输后，接收端计算接收报文中的校验，如果校验正确，则要求发送端继续发送后续报文，否则，要求发送端重发错误的报文。接收端的反馈信息分为两种 ACK 和 NACK，根据实现过程的不同，ARQ 大致可分为三种基本模型：停等型（SW, Stop and Wait）、选择重复型（SP, Selective Repeat）和返回 N 型（GBN, Go-Back-N）。

### 停等型（SW, Stop and Wait）

发送端发送数据报文后，等待接收端的反馈信息 ACK，如果 ACK 正确接收到，则继续发送后续报文，否则，在过了限定时间后，发送端重发数据报文。因此，停等型 ARQ 只能一个报文一个报文地发送，实现简单，但效率低。

### 选择重复型（SP, Selective Repeat）

发送端连续发送数据报文，接收端也逐一发送正确接收报的 ACK，如果在限定时间内没有接收到某个报文的 ACK，发送端则只重发出错报文，而后，继续前面连续发送的报文的工作。例如，当前发送端刚发送完报文 n，同时确定 n-3 报文错误，则只再重发报文 n-3，然后，继续发送报文 n+1。这就要求接收端有接收缓存，方法实现复杂，但效率高

### 返回 N 型（GBN, Go-Back-N）

返回 N 型的 ARQ 和选择重发一样，采用连续发送报文，一旦确定前面发送的某个报文出错，则发送端从出错报文的位置开始，重新发送。返回 N 型的实现较为简单，效率介于停等型和选择重复型之间。

**ARQ 差错控制方法实质就是以时延和吞吐量为代价换取传输的可靠性。**ARQ 会引入传输时延，限制了其在视频传输中的使用。但重传技术作为能够正确地恢复出错数据的方法，其作用是解码端的错误隐藏技术无法比拟的，可以彻底避免视频错误的传播。

当然，上面只是简单地描述了三种基本 ARQ 模型，在实时视频传输有着严格的时延要求，这一点与传统的非实时数据通信中显著不同。因此，在视频传输中 ARQ 往往进行了改进，使得整个系统在不影响实时的视频显示的情况下，重传出错的信息，阻止误码的传输，例如利用连续修正的误码扩展进行误码恢复的视频重传技术 RESCU。

为了尽量减少重传报文的数量，可以将 FEC 和 ARQ 结合起来，形成混合型 ARQ，混合型 ARQ 将 FEC 检错纠错功能和 ARQ 结合起来，最大限度的降低需要重传的比特流数量，提高网络的有效带宽。目前，以上各种 ARQ 的研究主要集中在如何根据信道的传输条件以及变化情况对 ARQ 进行优化计算等。

#### 9.2.2.4 无线信道的 IP 协议

未来的无线视频通信必然是开放分层的，需要和固定的 IP 网络进行互连互通。因此，制定能够在有线网络和无线网络共用的传输协议受到了愈来愈多的关注。这类传输协议是无法脱离现在广泛应用的 IP 协议。而无线信道上传输的帧往往小于最大 IP 报文，这势必会引入 IP 报文的分段和重装问题，即太大的 IP 报文必须在发送端分割成适当大小，以利于无线信道的传输，在接收端接收到这些小报文后，重新组装。这就产生一个不利因素：如果某个小报文损坏，原来大的 IP 报文就会完全丢弃，从而严重影响传输的丢包率。而且 IP 报文头（典型的 IP/UDP/RTP 头为 48 字节）也是一项不容忽视的额外开销。

### 9.2.3 无线视频通信的应用

移动终端传输视频流已经成为了第三代移动通信 3G 的主体业务之一。具体来说，无线视频通信业务服务可分为以下三类：

- 1) 应用于视频电话和视频会议的包交换会话服务（PCS）
- 2) 应用于直播和视频录像的包交换流媒体服务（PSS）
- 3) 多媒体信息服务（MMS）

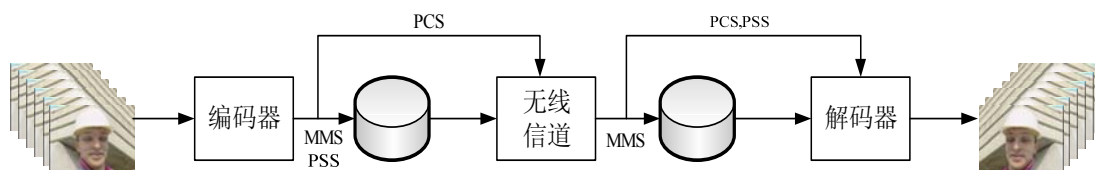


图 9.10 移动通信视频应用示意图

不同的视频应用具有不同的QoS等级和工作方式，如图9.10所示。

- 包交换会话服务（PCS）

编码、传输和解码实时双向同时进行，最小化会话服务中的端到端时延，确保视频和音频的同步；

- 包交换流媒体服务（PSS）

传输和解码同时进行，编码提前处理，一般把编码好的视频放入视频服务器的队列中；

- 多媒体信息服务（MMS）

编码，传输和解码三者完全分离。记录的视频信号进行离线编码和本地存储。然后可在任何时间进行发送，接受端在完全下载下来后才进行解码。

正如前面所说，无线网络的传输能力与日益增长的多媒体信息需求相比，有效带宽总是非常有限的，自然，用户希望他们使用无线网络所付出的代价与占用的带宽资源是成正比的。因此，低码流迎合了大众的需求，在移动环境的视频编码标准的成功取决于压缩效率。这样使能够提供目前最高压缩效率的视频编码标准 H.264/AVC 当仁不让地成为了无线系统的最佳候选标准。

除此之外，移动网络由于受地理环境、天气以及多用户冲突等因素的影响，造成网络波动较大。频道变化的频率高度地依赖环境、用户的位置、用户移动的速度和信号载波的频率。尽管可以采用诸如时空编码、多天线系统、快速控制、交叉等技术来改善信道的性能，但要正在为用户提供具有 QoS 保障的上述无线视频应用，因此，除了高压缩效率和合理的复杂度，适用于无线环境视频服务的视频编码标准必须具有较高的抗误码的能力。H.264/AVC 视频编码标准的制定着重考虑到这一点，相对于以往的视频编码标准，添加了多项错误控制技术，进一步改善了多种环境下的视频传输抗误码能力，满足了无线视频传输业务的需求。

## 9.2.4 H.264 无线通信中传输结构

多媒体视频在不同环境下的传输，不仅需要高效率的编码，同时编码后的视频应能容易地无缝集成到当前和未来的网络协议架构当中。鉴于 H.263 和 MPEG-4 标准制订过程中面临的网络应用的相关问题，H.264 在系统层面提出了一个全新的概念，定义了视频编码层 VCL（Video Coding Layer）和网络提取层 NAL（Network Adaptation Layer），如图 9.11 所示。

VCL 包括 VCL 编码器与 VCL 解码器，主要功能是采用运动补偿、变换编码、熵编码以及去方块滤波等多种技术，进行视频压缩数据的高效组织和表示。与以往的编码标准不同的是，引入了一系列新特性，使得 H.264 的编码压缩效率得到了极大提升。H.264 只明确地规定解码过程，对视频捕捉、预处理、编码等阶段的具体实现留有了充分的余地。VCL 也可根据当前的网络情况调整编码参数(如帧率等)来适应网络的变化。

NAL 由 NAL 编码器和 NAL 解码器组成，主要功能是定义了数据封装的格式，为 VCL 提供与网络无关的统一接口。数据承载在网络提取层单元（NALU）中，NALU 采用统一的数据格式，包含单个字节的包头信息和多个字节的数据。包头信息包含存储标志和类型标志。存储标志用于指示当前数据不属于被参考的帧，从而便于服务器根据网络的拥塞情况进行丢弃；类型标志用于指示图像数据的类型。NAL 支持众多基于包的有线/无线通信网络，诸如 H.320、MPEG-2 和 RTP/IP 等。

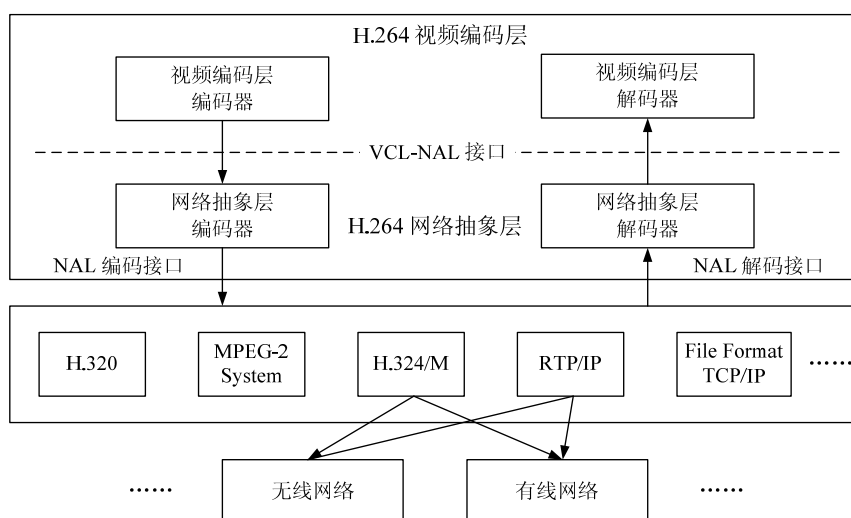


图9.11 H.264的传输结构

以移动通信中的视频传输而言，除了高效的视频压缩，视频数据传输如何适应无线信道和移动通信的特点，同样也是一个非常关键的问题。诸如 GSM/GPRS，UMTS 和 CDMA-2000 等移动网络，都设计了基于包的传输模式，以便为用户提供一种简单灵活基于 IP 协议的数据传输机制。第三代合作方案（3GPP）在其多媒体规范中也集成了几种多媒体编码器，为 3G 系统提供基本的视频服务。

图 9.12 给出了 NAL 单元通过 3GPP 用户平面协议栈被封装在 RTP/UDP/IP 的典型过程。NAL 首先作为 RTP 的有效载荷，加上 IP/UDP/RTP 的报文头，构成 IP 包；在头压缩（RoHC）后，此 IP/UDP/RTP 包被封装进 PPP 报文中，就变成了无线链路控制服务数据单元 RLC-SDU。RLC 有三种工作模式：透明、非确认和确认模式，为用户和数据提供分段和重传机制。其中，在透明和非确认模式，RLC 是单向的，对于出错协议数据单元 PDU，采用丢弃或标记的方法；在确认模式，RLC 是双向的，使用自动重传机制来确保数据的正确传输。

视频数据的大小是变化的，其封装的 RLC-SDU 长度自然也随之变化。协议数据单元 PDU 是数据传输的处理单位。因此，如果 RLC-SDU 比 RLC-PDU 大，SDU 就会被分成若干 PDU。

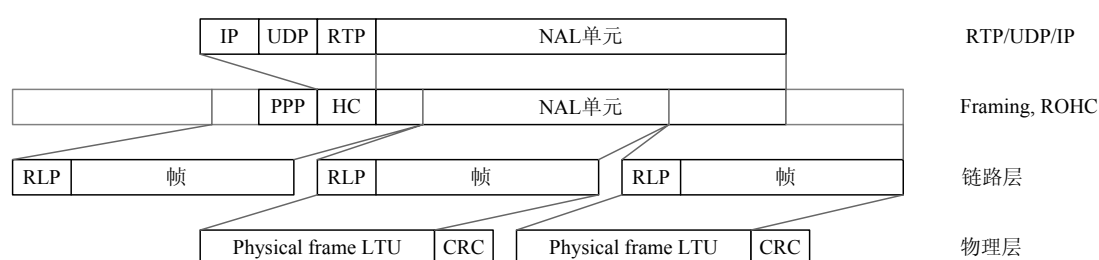


图 9.12 3GPP 用户平面协议栈封装过程

## 9.2.5 无线视频传输的鲁棒性研究

无线视频传输的鲁棒性在近几年无线视频应用的需求迅猛高涨起来后显得越来越突出。无线信道中存在着 Rayleigh 衰减和多用户干扰，会在传输位流中产生突发性错误(burst error)。视频压缩国际标推如 MPEG-x 和 H.26x 等都是基于宏块的压缩算法，通过运动估计/运动补偿(MP/MC)消除视频时间冗余，对差值图像进行离散余弦变换(DCT)变换消除空间冗余，对量化后的系数进行可变长编码(VLC)消除统计冗余。实践表明，通过上述方法，视频编码标准获得了极高的压缩效率。但压缩后的码流在无线信道上的传输仍然存在着一些棘手的问题，其中很突出的一点是：一方面，这些压缩后的码流对信道比特误码非常地敏感，而另一方面无线信道由于多径反射和衰落引入了大量的随机误码和突发误码，影响了码流的正常传输。尤其是当采用了 VLC 方案后，码流更加容易受到误码的影响，结果在解码端将失去与编码端的同步，导致在遇到下一个同步码字之前无法对 VLC 码字进行正确的解码；同时预测编码技术会将错误扩散到整个视频序列中，极大地降低重建图像的质量。因此，为了实现良好质量的视频传输，必须结合无线信道的传输特性，采取一定的容错措施。

文献[2]对视频通信中的容错算法进行了系统的回顾。根据在视频传输系统中位置的不同容错算法主要可分为基于编码器的容错算法、基于解码器的容错算法和基于反馈信道的容错算法。其中，

1) 基于编码器的容错算法，通过再编码比特流中添加冗余信息，这些冗余信息被添加在信源或信道编码器中，降低了编码的效率，增加了实现的复杂度，以换取编码的容错性能，大致包括：分层编码、多描述编码、独立分段编码、再同步编码和前向纠错编码 FEC 等；

2) 基于解码器的容错算法，是指利用被损坏的宏块与其相邻的宏块之间的相关性来完成恢复工作的，这部分工作包括错误检测和错误恢复。对于错误的检测，一般采用针对语法的检错和嵌入数据的检错；对于错误恢复，可采用时域和空域的错误隐藏方法。

3) 基于反馈信道的容错算法，指利用解码器获得误码信息，并通过反馈信道，传送给编码器进行误码处理的一种方式。主要包括：误码跟踪、有条件的 ARQ、帧内/帧间编码模式选择和参考图像选择模式等。



与此同时，随着无线视频应用的兴起，在信源编码器中，从视频码流结构上研究其抗误码性能，成为近两年来研究的一个热点。H.264/AVC 作为最新的视频编码标准，采取了一系列的切合实际的技术措施，提高了网络适应性，增强了数据的抗误码的鲁棒性，从而保证无线视频传输后的压缩视频的 QoS，下一节将进行详细论述。

### 9.3 H.264 视频编解码标准的错误恢复

与以往的视频编码标准不同的是，H.264/AVC 标准从系统层面定义了视频编码层（VCL，Video Coding Layer）和网络提取层（NAL，Network Abstraction Layer）。其中，**视频编码层独立于网络，主要包括核心压缩引擎和块、宏块和片的语法句法定义**。通过引入一系列新特性，不但使 H.264 的编码压缩效率提升了近 1 倍，而且多种错误恢复工具又增强了视频流的鲁棒性。**网络提取层的主要功能是定义数据的封装格式，把 VCL 产生的比特字符串适配到各种各样的网络和多元环境中。涉及片级别以上的语法定义**，包括独立片解码所要求的数据表示，类似以往视频压缩标准中的图像和头部顺序数据；防止竞争的编码；附加的增强信息以及编码片的比特字符串。

H.264 从框架结构上将 NAL 与 VCL 的分离，主要有两个目的：首先，可以定义 VCL 视频压缩处理与 NAL 网络传输机制的接口，这样允许视频编码层 VCL 的设计可以在不同的处理器平台进行移植，而与 NAL 层的数据封装格式无关；其二，VCL 和 NAL 都被设计成工作于不同的传输环境，异构的网络环境并不需要对 VCL 比特流进行重构和重编码。下面分别就 VCL 和 NAL 对于视频传输的 QoS 进行分析。

#### 9.3.1 H.264 的视频编码层的错误恢复

在 H.261、H.263、MPEG-1、MPEG-2、MPEG-4 中，许多错误恢复工具已经得到了很好的应用：图像分割的不同形式（片、块组）、I 模式宏块，片和图像的内插、参考图像选择（带有和不带反馈，图像级别、GOB/SLICE 或 MB 级别）、数据分割等。

H.264 标准继承以前视频编码标准中某些优秀的错误恢复工具，同时也改进和创新了多种错误恢复工具。这里主要介绍的 **H.264 的错误恢复工具：参数集；灵活的宏块排序；冗余片 RS。**

- **参数集**

参数集是 H.264 标准的一个新概念，**是一种通过改进视频码流结构增强错误恢复能力的方法**。H.264 的参数集又分为序列参数集和图像参数集，其中，序列参数集包括一个图像序列的所有信息，即两个 IDR 图像间的所有图像信息。图像参数集包括一个图像的所有分片的所有相关信息，包括图像类型、序列号等，解码时某些序列号的丢失可用来检验信息包的丢失与否。多个不同的序列和图像参数集存贮在解码器中，编码器通依据每个编码分片的头部的存贮位置来选择适当的参数集，图像参数集本身也包括使用的序列参数集参考信息。

众所周知，一些关键信息比特的丢失（如序列和图像的头信息）会造成解码的严重负面效应，而 H.264 把这些关键信息分离出来，凭借参数集的设计，确保在易出错的环境中能正确地传输。这种码流结构的设计无疑增强了码流传输的错误恢复能力。

参数集具体实现的方法也是多样化的：1) 通过带外传输，这种方式要求参数集通过可靠的协议，在首个片编码到达之前传输到解码器；2) 通过带内传输，这需要为参数集提供更高级别的保护，例如发送复制包来保证至少有一个到达目标；3) 在编码器和解码器采用硬件处理参数集。

- **片、片组和 FMO**

一幅图像由若干片组成，每片包含一系列的宏块（MB）。MB 的排列可按光栅扫描顺序，也可不按扫描顺序。每个片独立解码，不同片的宏块不能用于自身片中作预测参考。因此，片的设置不会造成误码扩散。如图 9.13 所示，片 0、片 1 和片 2 中的宏块相互之间不能作为预测参考。

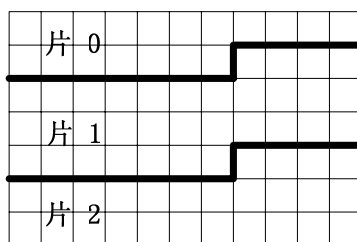


图 9.13 不采用 FMO 的片划分

灵活的宏块排序 FMO 是 H.264 的一大特色，适用于 H.264 的基本档次和扩展档次的应用。FMO 通过宏块分配映射技术，把每个宏块分配到不按扫描顺序的片中，图像内部预测机制，例如帧内预测或运动矢量预测，仅允许用同一组里的空间相邻的宏块。FMO 模式划分图像的模式各种各样，重要的有棋盘模式、矩形模式等。当然 FMO 模式也可以使一帧中的宏块顺序分割，使得分割后的片的大小小于无线网络的 MTU 尺寸，经过 FMO 模式分割后的图像数据分开进行传输。

以图 9.14 所示，所有的 MB 被分成了片组 0 和片组 1，相应地分别采用黑色和白色表示。当白片丢失时，因为其周围的宏块都属于其他片的宏块，利用邻域相关性，黑片宏块的某种加权可用来代替白片相应宏块。这种错误隐藏机制可以明显提高抗误码性能。实验证明，在 CIF 图象的视频会议中，在丢包率达 10% 时，视频失真低到需要训练有素的眼睛才能识别。使用 FMO 的代价是稍微降低了编码效率，（因为它打破了原先非邻居 MB 之间的预测），而且在高度优化的环境中，有比较高的延迟。

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23

图 9.14 FMO 棋盘分片例图

### ● 数据分割

通常情况下，一个宏块的数据是存放在一起而组成片的，数据划分使得一个片中的宏块数据重新组合，把宏块语义相关的数据组成一个划分，由划分来组装片。H.264 视频编码标准使用了三种不同类型的数据分割：

#### 1) A 型分割

A 型分割是头信息划分，包括宏块类型、量化参数和运动矢量，这个信息是最重要的。

#### 2) B 型分割

B 型分割是帧内信息划分，包括帧内 CBPs 和帧内系数，帧内信息可以阻止错误的传播，该型数据分割要求给定分片的 A 型分割有效，相对于帧间信息，帧内信息能更好地阻止漂移效应，因此它比帧间分割更为重要。

#### 3) C 型分割

C 型分割是帧间信息划分，包括帧间 CBPs 和帧间系数，一般情况下它是编码分片的最大分区。帧间分割是最不重要的，它的使用要求 A 型分割有效。

当使用数据分割时，源编码器把不同的类型的分割安排在三个不同的缓冲器中，同时分片的尺寸必须进行调整以保证小于 MTU 长度，因此是编码器而不是 NAL 来实现数据分割。在解码器上，所有分割用于信息重建。这样，如果帧内或帧间信息丢失了，有效的帧头信息仍然能用来提高错误隐藏效率，即有效的宏块类型和运动矢量，保留了宏块的基本特征，从而仍可获得一个相当高的信



息重构质量，而仅仅丢失了细节信息。

- 冗余片方法

H.264 中的参考图像的选择与以前在 H.263 一样，在基于反馈的系统中，解码器接收到丢失或被破坏的图像信息时，选择参考图像序列中正确的参考宏块，来进行错误恢复；而对于无反馈的系统，H.264 提出了冗余分片编码。

冗余分片允许编码器把在同一个码流中添加同一 MB 的一个或更多冗余表示。需要注意的是这些冗余片的编码参数与非冗余片的编码参数不同，例如主片可用低 QP（高质量）来编码，而冗余信息中能用一个高 QP（低质量）的方式来编码，这样质量粗糙一些但码率更低。解码器在重构时，首先使用主片，如果它可用就抛弃冗余片；而如主片丢失（比如因为包的丢失）冗余片也能被用于重构。冗余片主要用于支持高误码的移动环境。

- 帧内编码

H.264 中帧内编码大体上类似于以往的视频编码标准，但也进行了重要的改进，主要体现在：

1) H.264 中帧内预测宏块的参考宏块可以是帧间编码宏块，帧内预测宏块并不像 H.263 中的帧内编码一样，而采用预测的帧内编码比非预测的帧内编码有更好的编码效率，但降低了帧内编码的重同步性能，可以通过设置限制帧内预测标记来恢复这一性能。

2) 只包含帧内宏块的片有两种，一种是帧内片(I Slice)，一种是立即刷新片(IDR Slice)，立即刷新片需存在于立即刷新图像(IDR Picture)中。与短期参考图像相比，立即刷新图像有更强壮的重同步性能。

为了更适用无线 IP 网络环境中的应用，H.264 通过采用率失真优化编码和设置帧内预测标志，来提高帧内图像的重同步性能，

### 9.3.1 H.264 的网络提取层的错误恢复

NAL 支持众多基于包的有线/无线通信网络，诸如 H.320、MPEG-2 和 RTP/IP 等。但目前，绝大部分的视频应用所采用的网络协议层次是 RTP/UDP/IP，因此在下面的描述中主要基于这个传输框架。下面首先分析 NAL 层的基本处理单元 NALU 以及它的网络封装、分割和合并的方法，

- NAL 单元

每个 NAL 单元是一个一定语法元素的可变长字节字符串，包括包含一个字节的头信息（用来表示数据类型），以及若干整数字节的负荷数据。一个 NAL 单元可以携带一个编码片、A/B/C 型数据分割或一个序列或图像参数集。

NAL 单元的头信息结构如图 2 所示，NAL 单元按 RTP 序列号按序传送。其中，T 为负荷数据类型，占 5 个比特；R 为重要性指示位，占 2 个比特；最后的 F 为禁止位，占 1 个比特。具体如下：

0	1	2	3	4	5	6	7
T					R		F

图 9.15 NAL 单元头结构示意图

1) NALU 类型位

可以表示 NALU 的 32 种中不同类型特征，类型 1~12 是 H.264 的定义的，类型 24~31 是用于 H.264 以外的，RTP 负荷规范使用这其中的一些值来定义包聚合和分裂，其他值为 H.264 保留。

2) 重要性指示位

用于在重构过程中标记一个 NAL 单元的重要性，值越大，越重要。值为 0 表示这个 NAL 单元没有用于预测，因此可被解码器抛弃而不会有错误扩散，值高于 0 表示此 NAL 单元要用于无漂移重构，且值越高，对此 NAL 单元丢失的影响越大。

3) 禁止位

编码中默认置为 0，当网络识别此单元中存在比特错误时，可将其置为 1，以便接收方丢掉该单元，主要用于适应不同种类的网络环境(比如有线无线相结合的环境)。例如对于从无线到有线的网关，一边是无线的非 IP 协议环境，一边是有线网络的无比特错误的环境。假设一个 NAL 单元到达无线那边时，校验和检测失败，网关可以选择从 NAL 流中去掉这个 NALU 单元，也可以把已知被破坏的 NAL 单元前传给接收端。在这种情况下，智能的解码器将尝试重构这个 NAL 单元（已知它可能包含比特错误）。而非智能的解码器将简单的抛弃这个 NAL 单元。

NAL 单元结构规定了用于面向分组或用于流的传输子系统的通用格式。在 H.320 和 MPEG-2 系统中，NAL 单元的流应该在 NAL 单元边界内，每个 NAL 单元前加一个 3 个字节的起始前缀码。在分组传输系统中，NAL 单元由系统的传输规程确定帧界，因此不需要上述的起始前缀码。

一组 NAL 单元被称为一个接入单元，定界后加上定时信息（SEI），形成基本编码图像。该基本编码图像（PCP）由一组已编码的 NAL 单元组成，其后是冗余编码图像（RCP），它是 PCP 同一视频图像的冗余表示，用于解码中 PCP 丢失情况下恢复信息。如果该编码视频图像是编码视频序列的最后一幅图像，应出现序列 NAL 单元的 end，表示该序列结束。一个图像序列只有一个序列参数组，并被独立解码。如果该编码图像是整个 NAL 单元流的最后一幅图像，则应出现流的 end。于是一个接入单元的组成见图 9.16。

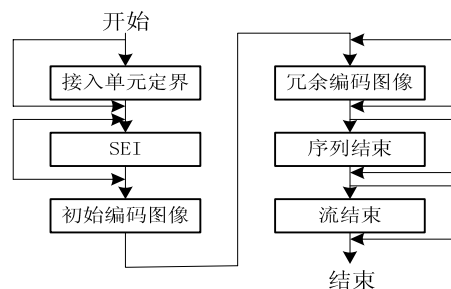


图 9.16 接入单元头结构

如图 9.16 所示，H.264 采用上述严格的接入单元，不仅使 H.264 可自适应于多种网络，而且进一步提高其抗误码能力。序列号的设置可发现丢的是哪一个 VCL 单元。冗余编码图像使得即使基本编码图像丢失，仍可得到较“粗糙”图像。

#### ● H.264 中的 RTP

上面阐述了 NAL 单元的结构和实现，这里要详细讨论 RTP 的载荷规范和抗误码性能。**RTP 可通过发送冗余信息来减少接收端的丢包率，会增加时延，与冗余片不同的是它增加的冗余信息是个别重点信息的备份，适合于的非平等保护机制。**相应的多媒体传输规范有：

a) 分组复制多次重发，发送端对最重要的比特信息分组进行复制重发，使得保证接收端能至少正确接收到一次，同时接收端要丢弃已经正确接收的分组的多余备份。

b) 基于分组的前向纠错，对被保护的分组进行异或运算，将运算结果作为冗余信息发送到接收方。由于时延，不用于对话型应用，可用于流媒体。

c) 音频冗余编码，可保护包括视频在内的任何数据流。每个分组由头标、载荷以及前一分组的载荷组成，H.264 中可与数据分割一起使用。

RTP 的封装规范总结如下：

a) 额外开销要少，使 MTU 尺寸在 100~64k 字节范围都可以；

b) 易于区分分组的重要性，而不必对分组内的数据解码；

c) 载荷规范应当保证不用解码就可识别由于其他的比特丢失而造成的分组不可解码；

d) 支持将 NALU 分割成多个 RTP 分组；

e) 支持将多个 NALU 汇集在一个 RTP 分组中。

H.264 采用了简单打包的方案，即一个 RTP 分组里放入一个 NALU，将 NALU(包括同时作为载

荷头标的 NALU 头)放入 RTP 的载荷中, 设置 RTP 头标值。理想情况下, VCL 不会产生超过 MTU 尺寸的 NAL 单元, 来避免 IP 层的分拆。在接收端, 通过 RTP 序列信息识别复制包并丢弃, 取出有效 RTP 包里的 NAL 单元。基本档次和扩展档次允许片的无序解码, 这样在抖动缓存中就不必对包重新排序。在使用主档次时(不允许片的乱序), 要通过 RTP 序列信息来对包重新排序, 解码顺序号(DON)的概念现正在 IETF 的讨论中。

存在如下情况, 例如当使用内容预编码时, 编码器不了解底层网络的 MTU 大小, 将产生许多大于 MTU 尺寸的 NALU。这就需要涉及 NALU 分割和合并。

#### 1) NALU 的分割

虽然 IP 层的分割可以使数据块小于 64 千字节, 但无法在应用层实现保护, 从而降低了非平等保护方案的效果。由于 UDP 数据包小于 64 千字节, 而且一个片的长度对某些应用场合来说太小, 所以应用层打包是 RTP 打包方案的一部分。目前的拆分方案正在 IETF 的讨论之中, 大致具有一下特点:

- a) NALU 的分块以按 RTP 次序号升序传输;
- b) 能够标记第一个和最后一个 NALU 分块;
- c) 可以检测丢失的分块。

#### 2) NALU 的合并

一些 NALU 如 SEI、参数集等非常小, 将它们合并在一起有利于减少头标开销。现有的两种集合分组:

- a) 单一时间集合分组(STAP), 按时间戳进行组合, 一般用于低延迟环境;
- b) 多时间集合分组(MTAP), 不同时间戳也可以组合, 一般用于高延迟环境, 比如流应用。

本节针对最新推出的视频编解码标准 H.264 的抗误码性能进行了单独阐述, 可以看到 H.264 除了拥有面向高效编码的特性, 还引入了一些新工具用于提高错误恢复能力。特别是, 参数集、NAL 上的 NALU 的概念、灵活的宏块排序 FMO、数据分割以及帧内编码等都极大地复杂网络环境下的抗误码能力。同时, 详细介绍了于视频比特流传输密切相关的 RTP 封装规范, 与 H.264 的 NAL 紧密结合, 提供了对数据封装的指导。通过附加了一些传输层的低开销机制来用于 NALU 包的高效拆分和聚合。当联合使用这些工具时, 可以达到更高的性能, 在因特网和恶劣的无线网络上进行高质量的视频压缩也将最终成为现实。

## 参考文献

1. Y. Wang and Q. Zhu, "Error control and concealment for video communication: a review," *Proceedings of the IEEE*, vol. 86, pp. 974-997, May 1998
2. Stephan Wenger, "H.264/AVC Over IP," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 645-656, July 2003.
3. J. Wen and J. Villasenor, "A class of reversible variable length codes for robust image and video coding", *Proc. 1997 IEEE International Conf. On Image Proc.*, Santa Barbara, CA Oct. 1997.
4. J. Wen and J. Villasenor, "Reversible variable length codes for robust image and video transmission". *1997 Asilomar Conference*, Pacific Grove, CA, Nov. 1997.
5. D. W. Redmill and N. G. Kingsbury, "The EREC: An error resilient technique for coding variablelength blocks of data," *IEEE Trans. Image Proc.*, vol. 5, no. 4, pp. 565-574, Apr. 1996.
6. P. Haskell and D. Messerschmitt, "Resynchronization of motion compensated video affected by ATM cell loss," in *ICASSP-92*, San Francisco, CA, USA, Mar. 1992, vol.3, pp. 545-548
7. G. Côté, S. Shirani and F. Kossentini: "Optimal mode selection and synchronization for robust video communications over error prone networks" *IEEE Journal on Selected Areas in Communications*, May 1999, Submitted.
8. S. Wenger, "Video redundancy coding in H.263+," *Proc. AVSPN*, (Aberdeen, UK), Sept. 1997.
9. S. Wenger, G. Knorr, J. Ott, F. Kossentini: "Error resilience support in H.263+", *IEEE Trans. on circuit and System for Video Technology*, vol. 8, no. 6 pp. 867-877, Nov. 1998.
10. K. N. Ngan and C. W. Yap, "Combined source-channel video coding," in *Signal Recovery Techniques for Image and Video Compression and Transmission*, A. K. Katsaggelos and N. P. Galatsanos, editors, ch. 9, pp. 269-297, Kluwer Academic Publishers, 1998
11. L. Kondi, F. Ishtiaq, and A. K. Katsaggelos, "Joint source-channel coding for scalable video," *Proc. 2000 SPIE Conf. On Visual Communications and Image Processing*, San Jose, CA, Jan. 2000.
12. V. A. Vaishampayan, "Design of multiple description scalar quantizers," *IEEE Trans. Info. Theo.*, vol. 39, pp. 821-834, May 1993.
13. A. Ingle and V. A. Vaishampayan, "DPCM system design for diversity systems with applications to packetized speech," *IEEE Trans. Speech and Audio Processing*, vol. 3, pp. 48-57, Jan. 1995.
14. Y. Wang, M. Orchard, and A. Reibman, "Optimal pairwise correlating transforms for multiple description coding", *IEEE Int. Conf. Image Proc. (ICIP98)*, (Chicago, IL), Oct, 1998. Vol. 1, pp. 679-683.
15. V. K. Goyal, J. Kovacevic, R. Arian and M. Vetterli, "Multiple description transform coding of images," *IEEE Int. Conf. Image Proc. (ICIP98)*, (Chicago, IL), Oct, 1998. Vol. 1, pp. 674-678.
16. X. Yang and K. Ramchandran, "Optimal multiple description subband coding," *IEEE Int. Conf. Image Proc. (ICIP98)*, (Chicago, IL), Oct, 1998. Vol. 1, pp. 684-658.
17. Y. Wang and D. Chung, "Non-hierarchical signal decomposition and maximally smooth

- reconstruction for wireless video transmission,” in Goodman and Raychaudhuri, eds., *Mobile Multimedia Communications*, pp. 285-292. Plenum Press. 1997.
18. Q. Zhu and Y. Wang, “Error concealment in visual communications”, in A. R. Reibman and M. T. Sun, eds. *Compressed Video over Networks*. Marcel Dekker, Inc. To appear 2000.
  19. S. Aign, “Error concealment for MPEG-2 video,” in *Signal Recovery Techniques for Image and Video Compression and Transmission*, A. K. Katsaggelos and N. P. Galatsanos, editors, ch. 8, pp. 235-268, Kluwer Academic Publishers, 1998.
  20. M.C. Hong, L. Kondi, H. Scwab, and A. K. Katsaggelos, “Video Error Concealment Techniques,” *Signal Processing: Image Communications*, special issue on *Error Resilient Video*, 14(6-8), pp. 437-492, 1999.
  21. Q.-F. Zhu, Y. Wang, and L. Shaw, “Coding and cell loss recovery for DCT-based packet video,” *IEEE Trans. CAS for Video Tech.*, vol. 3, no. 3, pp. 248-258, June 1993.
  22. H. Sun and W. Kwok, “Concealment of damaged block transform coded images using projections onto convex sets,” *IEEE Trans. Image Proc.*, vol. 4, no. 4, pp. 470-477, Apr. 1995.
  23. G.-S. Yu, M. M.-K. Liu, and M. W. Marcellin, “POCS-based error concealment for packet video using multiframe overlap information,” *IEEE Trans. CAS for Video Tech.*, vol. 8, no. 4, pp. 422-434, Aug. 1998.
  24. H. Sun, K. Challapali, and J. Zdepski, “Error concealment in digital simulcast AD-HDTV decoder,” *IEEE Trans. Consumer Electronics*, vol. 38, no. 3, pp. 108-117, Aug. 1992.
  25. W.-M. Lam, A. R. Reibman, and B. Liu, “Recovery of lost or erroneously received motion vectors”, *Proc. ICASSP’93*, Minneapolis, Apr. 1993, pp. V417-V420.
  26. A. Narula and J. S. Lim, “Error concealment techniques for an all-digital high-definition television system,” *SPIE Visual Commun. And Image Proc.*, Cambridge, MA, 1993, pp. 304-315.
  27. B. Girod and N. Färber: "Feedback-based error control for mobile video transmission", *Proc. IEEE, Special Issue on Video for Mobile Multimedia*, vol. 87, pp. 1707-1723, Oct. 1999.
  28. Y. Tomita, T. Kimura and T. Ichikawa, “Error resilient modified inter-frame coding system for limited reference picture memories,” *Int. Picture Coding Symp. (PCS)*, Berlin, Germany, Sept. 1997, pp.743-748.
  29. T. Stockhammer, M. Hannuksela, and T. Wiegand, “H.264/AVC in wireless environments,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 657–673, July 2003.
  30. Yao Wang, Jorn Ostermann, Ya-Qin Zhang, *Video Processing and Communication*, Pearson Education, 2002.
  31. Bernd Girod and Niko Farkber, *Compressed Video Over Networks*, 1999.10
  32. 毕厚杰, 陈启美, 方晖, *IP宽带通信网络技术*, 北京邮电大学出版社, 2004.4
  33. 毕厚杰, *H.264/AVC视频编码标准的技术特点和评价*, 世界电信, 2004.7
  34. 沈兰荪, 田冻, *无线视频传输技术的发展*, 电子技术应用, 2001.1
  35. 吴伟陵, *移动通信中的关键技术*, 北京邮电大学出版社, 2000.11
  36. 冯秀波, 谢剑英, *无线视频传输容错算法研究新进展*, 通信学报, 2003.11
  37. Hang Liu, Haruo Ma, Error control schemes for networks: An overview, *Journal on Mobile Networks and Application*, 1997.2, p167-182

## 术语及英文解释

- **access unit**: A set of *NAL units* always containing a *primary coded picture*. In addition to the *primary coded picture*, an access unit may also contain one or more *redundant coded pictures* or other *NAL units* not containing *slices* or *slice data partitions* of a *coded picture*. The decoding of an access unit always results in a *decoded picture*.
- **AC transform coefficient**: Any *transform coefficient* for which the *frequency index* in one or both dimensions is non-zero.
- **adaptive binary arithmetic decoding process**: An *entropy decoding process* that recovers the values of *bins* from a *bitstream* produced by an *adaptive binary arithmetic encoding process*.
- **adaptive binary arithmetic encoding process**: An *entropy encoding process*, not normatively specified in this Recommendation | International Standard, that codes a sequence of *bins* and produces a *bitstream* that can be decoded using the *adaptive binary arithmetic decoding process*.
- **arbitrary slice order**: A *decoding order* of *slices* in which the *macroblock address* of the first *macroblock* of some *slice* of a *picture* may be smaller than the *macroblock address* of the first *macroblock* of some other preceding *slice* of the same *coded picture*.
- **B slice**: A *slice* that may be decoded using *intra prediction* from decoded samples within the same *slice* or *inter prediction* from previously-decoded *reference pictures*, using at most two *motion vectors* and *reference indices* to *predict* the sample values of each *block*.
- **bin**: One bit of a *bin string*.
- **binarization**: The set of intermediate binary representations of all possible values of a *syntax element*.
- **binarization process**: A unique mapping process of possible values of a *syntax element* onto a set of *bin strings*.
- **bin string**: A string of *bins*. A bin string is an intermediate binary representation of values of *syntax elements*.
- **bi-predictive slice**: See *B slice*.
- **bitstream**: A sequence of bits that forms the representation of *coded pictures* and associated data forming one or more *coded video sequences*. Bitstream is a collective term used to refer either to a *NAL unit stream* or a *byte stream*.
- **block**: An  $M \times N$  ( $M$ -column by  $N$ -row) array of samples, or an  $M \times N$  array of *transform coefficients*.
- **bottom field**: One of two *fields* that comprise a *frame*. Each row of a *bottom field* is spatially located immediately below a corresponding row of a *top field*.
- **bottom macroblock (of a macroblock pair)**: The *macroblock* within a *macroblock pair* that contains the samples in the bottom row of samples for the *macroblock pair*. For a *field*

*macroblock pair*, the bottom macroblock represents the samples from the region of the *bottom field* of the *frame* that lie within the spatial region of the *macroblock pair*. For a *frame macroblock pair*, the bottom macroblock represents the samples of the *frame* that lie within the bottom half of the spatial region of the *macroblock pair*.

- **broken link**: A location in a *bitstream* at which it is indicated that some subsequent *pictures* in *decoding order* may contain serious visual artefacts due to unspecified operations performed in the generation of the *bitstream*.
- **byte**: A sequence of 8 bits, written and read with the most significant bit on the left and the least significant bit on the right. When represented in a sequence of data bits, the most significant bit of a byte is first.
- **byte-aligned**: A bit in a *bitstream* is byte-aligned when its position is an integer multiple of 8 bits from the first bit in the *bitstream*.
- **byte stream**: An encapsulation of a *NAL unit stream* containing *start code prefixes* and *NAL units* as specified in Annex B.
- **category**: A number associated with each *syntax element*. The category is used to specify the allocation of *syntax elements* to *NAL units* for *slice data partitioning*. It may also be used in a manner determined by the application to refer to classes of *syntax elements* in a manner not specified in this Recommendation | International Standard.
- **chroma**: An adjective specifying that a sample array or single sample is representing one of the two colour difference signals related to the primary colours. The symbols used for a chroma array or sample are Cb and Cr.
  - NOTE - The term chroma is used rather than the term chrominance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term chrominance.
- **coded field**: A *coded representation* of a *field*.
- **coded frame**: A *coded representation* of a *frame*.
- **coded picture**: A *coded representation* of a *picture*. A coded picture may be either a *coded field* or a *coded frame*. Coded picture is a collective term referring to a *primary coded picture* or a *redundant coded picture*, but not to both together.
- **coded picture buffer (CPB)**: A first-in first-out buffer containing *access units* in *decoding order* specified in the *hypothetical reference decoder* in Annex C.
- **coded representation**: A data element as represented in its coded form.
- **coded video sequence**: A sequence of *access units* that consists, in decoding order, of an *IDR access unit* followed zero or more non-IDR *access units* including all subsequent *access units* up to but not including any subsequent *IDR access unit*.
- **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that make up a *field* or *frame*.
- **complementary field pair**: A collective term for a *complementary reference field pair* or a *complementary non-reference field pair*.

- **complementary non-reference field pair:** Two *non-reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* of opposite parity where the first *field* is not already a paired *field*.
- **complementary reference field pair:** Two *reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* and share the same value of *frame number*, where the second *field* in *decoding order* is not an *IDR picture* and does not include a *memory\_management\_control\_operation syntax element* equal to 5.
- **context variable:** A variable specified for the *adaptive binary arithmetic decoding process* of a *bin* by an equation containing recently decoded *bins*.
- **DC transform coefficient:** A *transform coefficient* for which the *frequency index* is zero in all dimensions.
- **decoded picture:** A *decoded picture* is derived by decoding a *coded picture*. A *decoded picture* is either a *decoded frame*, or a *decoded field*. A *decoded field* is either a *decoded top field* or a *decoded bottom field*.
- **decoded picture buffer (DPB):** A buffer holding *decoded pictures* for reference, output reordering, or output delay specified for the *hypothetical reference decoder* in Annex C.
- **decoder:** An embodiment of a *decoding process*.
- **decoding order:** The order in which *syntax elements* are processed by the *decoding process*.
- **decoding process:** The process specified in this Recommendation | International Standard that reads a *bitstream* and produces *decoded pictures*.
- **direct prediction:** An *inter prediction* for a *block* for which no *motion vector* is decoded. Two *direct prediction* modes are specified that are referred to as *spatial direct prediction* and *temporal prediction* mode.
- **decoder under test (DUT):** A *decoder* that is tested for conformance to this Recommendation | International Standard by operating the *hypothetical stream scheduler* to deliver a conforming *bitstream* to the *decoder* and to the *hypothetical reference decoder* and comparing the values and timing of the output of the two *decoders*.
- **emulation prevention byte:** A byte equal to 0x03 that may be present within a *NAL unit*. The presence of emulation prevention bytes ensures that no sequence of consecutive byte-aligned bytes in the *NAL unit* contains a *start code prefix*.
- **encoder:** An embodiment of an *encoding process*.
- **encoding process:** A process, not specified in this Recommendation | International Standard, that produces a *bitstream* conforming to this Recommendation | International Standard.
- **field:** An assembly of alternate rows of a *frame*. A *frame* is composed of two *fields*, a *top field* and a *bottom field*.
- **field macroblock:** A *macroblock* containing samples from a single *field*. All *macroblocks* of a *coded field* are field macroblocks. When *macroblock-adaptive frame/field decoding* is in use, some macroblocks of a *coded frame* may be field macroblocks.



- **field macroblock pair:** A *macroblock pair* decoded as two *field macroblocks*.
- **field scan:** A specific sequential ordering of *transform coefficients* that differs from the *zig-zag* scan by scanning columns more rapidly than rows. Field scan is used for *transform coefficients* in *field macroblocks*.
- **flag:** A variable that can take one of the two possible values 0 and 1.
- **frame:** A *frame* contains an array of luma samples and two corresponding arrays of chroma samples. A *frame* consists of two *fields*, a *top field* and a *bottom field*.
- **frame macroblock:** A *macroblock* representing samples from two *fields* of a *coded frame*. When *macroblock-adaptive frame/field decoding* is not in use, all *macroblocks* of a *coded frame* are frame *macroblocks*. When *macroblock-adaptive frame/field decoding* is in use, some *macroblocks* of a *coded frame* may be frame *macroblocks*.
- **frame macroblock pair:** A *macroblock pair* decoded as two *frame macroblocks*.
- **frequency index:** A one-dimensional or two-dimensional index associated with a *transform coefficient* prior to an *inverse transform* part of the *decoding process*.
- **hypothetical reference decoder (HRD):** A hypothetical *decoder* model that specifies constraints on the variability of conforming *NAL unit streams* or conforming *byte streams* that an encoding process may produce.
- **hypothetical stream scheduler (HSS):** A hypothetical delivery mechanism for the timing and data flow of the input of a *bitstream* into the *hypothetical reference decoder*. The HSS is used for checking the conformance of a *bitstream* or a *decoder*.
- **I slice:** A *slice* that is decoded using *prediction* only from decoded samples within the same *slice*.
- **instantaneous decoding refresh (IDR) access unit:** An *access unit* in which the *primary coded picture* is an *IDR picture*.
- **instantaneous decoding refresh (IDR) picture:** A *coded picture* containing only *slices* with *I* or *SI slice types* that causes the *decoding process* to mark all *reference pictures* as "unused for reference" immediately after decoding the *IDR picture*. After the decoding of an *IDR picture* all following *coded pictures* in *decoding order* can be decoded without *inter prediction* from any *picture* decoded prior to the *IDR picture*. The first *picture* of each *coded video sequence* is an *IDR picture*.
- **inter coding:** Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *inter prediction*.
- **inter prediction:** A *prediction* derived from decoded samples of *reference pictures* other than the current *decoded picture*.
- **intra coding:** Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *intra prediction*.
- **intra prediction:** A *prediction* derived from the decoded samples of the same *decoded slice*.
- **intra slice:** See *I slice*.

- **inverse transform:** A part of the *decoding process* by which a set of *transform coefficients* are converted into spatial-domain values, or by which a set of *transform coefficients* are converted into *DC transform coefficients*.
- **layer:** One of a set of syntactical structures in a non-branching hierarchical relationship. Higher layers contain lower layers. The coding layers are the *coded video sequence*, *picture*, *slice*, and *macroblock* layers.
- **level:** A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this Recommendation | International Standard. The same set of levels is defined for all *profiles*, with most aspects of the definition of each level being in common across different *profiles*. Individual implementations may, within specified constraints, support a different level for each supported *profile*. In a different context, *level* is the value of a *transform coefficient* prior to *scaling*.
- **list 0 (list 1) motion vector:** A *motion vector* associated with a *reference index* pointing into *reference picture list 0 (list 1)*.
- **list 0 (list 1) prediction:** *Inter prediction* of the content of a *slice* using a *reference index* pointing into *reference picture list 0 (list 1)*.
- **luma:** An adjective specifying that a sample array or single sample is representing the monochrome signal related to the primary colours. The symbol used for luma is Y.
  - NOTE – The term luma is used rather than the term luminance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term luminance.
- **macroblock:** A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples. The division of a *slice* or a *macroblock pair* into macroblocks is a *partitioning*.
- **macroblock-adaptive frame/field decoding:** A *decoding process* for *coded frames* in which some *macroblocks* may be decoded as *frame macroblocks* and others may be decoded as *field macroblocks*.
- **macroblock address:** When *macroblock-adaptive frame/field decoding* is not in use, a macroblock address is the index of a macroblock in a *macroblock raster scan* of the *picture* starting with zero for the top-left *macroblock* in a *picture*. When *macroblock-adaptive frame/field decoding* is in use, the macroblock address of the *top macroblock* of a *macroblock pair* is two times the index of the *macroblock pair* in a *macroblock pair raster scan* of the *picture*, and the macroblock address of the *bottom macroblock* of a *macroblock pair* is the macroblock address of the corresponding *top macroblock* plus 1. The macroblock address of the *top macroblock* of each *macroblock pair* is an even number and the macroblock address of the *bottom macroblock* of each *macroblock pair* is an odd number.
- **macroblock location:** The two-dimensional coordinates of a *macroblock* in a *picture* denoted by ( *x*, *y* ). For the top left *macroblock* of the *picture* ( *x*, *y* ) is equal to ( 0, 0 ). *x* is incremented by 1 for each *macroblock* column from left to right. When *macroblock-adaptive frame/field decoding* is not in use, *y* is incremented by 1 for each *macroblock* row from top to bottom. When *macroblock-adaptive frame/field decoding* is in use, *y* is incremented by 2 for each *macroblock pair* row from top to bottom, and is incremented by an additional 1 when a macroblock is a *bottom macroblock*.

- **macroblock pair:** A pair of vertically contiguous *macroblocks* in a *frame* that is coupled for use in *macroblock-adaptive frame/field decoding* processing. The division of a *slice* into macroblock pairs is a *partitioning*.
- **macroblock partition:** A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction*.
- **macroblock to slice group map:** A means of mapping *macroblocks* of a *picture* into *slice groups*. The macroblock to slice group map consists of a list of numbers, one for each coded *macroblock*, specifying the *slice group* to which each coded *macroblock* belongs.
- **map unit to slice group map:** A means of mapping *slice group map units* of a *picture* into *slice groups*. The map unit to slice group map consists of a list of numbers, one for each *slice group map unit*, specifying the *slice group* to which each coded *slice group map unit* belongs.
- **memory management control operation:** Seven operations that control *reference picture marking*.
- **motion vector:** A two-dimensional vector used for *inter prediction* that provides an offset from the coordinates in the *decoded picture* to the coordinates in a *reference picture*.
- **NAL unit:** A syntax structure containing an indication of the type of data to follow and bytes containing that data in the form of an *RBSP* interspersed as necessary with *emulation prevention bytes*.
- **NAL unit stream:** A sequence of *NAL units*.
- **non-paired field:** A collective term for a *non-paired reference field* or a *non-paired non-reference field*.
- **non-paired non-reference field:** A decoded *non-reference field* that is not part of a *complementary non-reference field pair*.
- **non-paired reference field:** A decoded *reference field* that is not part of a *complementary reference field pair*.
- **non-reference field:** A *field* coded with `nal_ref_idc` equal to 0.
- **non-reference frame:** A *frame* coded with `nal_ref_idc` equal to 0.
- **non-reference picture:** A *picture* coded with `nal_ref_idc` equal to 0. A *non-reference picture* is not used for *inter prediction* of any other *pictures*.
- **opposite parity:** The *opposite parity* of *top* is *bottom*, and vice versa.
- **output order:** The order in which the *decoded pictures* are output from the *decoded picture buffer*.
- **P slice:** A *slice* that may be decoded using *intra prediction* from decoded samples within the same *slice* or *inter prediction* from previously-decoded *reference pictures*, using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*.
- **parameter:** A syntax element of a *sequence parameter set* or a *picture parameter set*. Parameter is also used as part of the defined term *quantisation parameter*.

- **parity:** The *parity* of a *field* can be *top* or *bottom*.
- **partitioning:** The division of a set into subsets such that each element of the set is in exactly one of the subsets.
- **picture:** A collective term for a *field* or a *frame*.
- **picture order count:** A variable having a value that is non-decreasing with increasing *picture* position in output order relative to the previous *IDR picture* in *decoding order* or relative to the previous *picture* containing the *memory management control operation* that marks all *reference pictures* as “unused for reference”.
- **prediction:** An embodiment of the *prediction process*.
- **prediction process:** The use of a *predictor* to provide an estimate of the sample value or data element currently being decoded.
- **predictive slice:** See P slice.
- **predictor:** A combination of previously decoded sample values or data elements used in the *decoding process* of subsequent sample values or data elements.
- **primary coded picture:** The coded representation of a *picture* to be used by the *decoding process* for a bitstream conforming to this Recommendation | International Standard. The primary coded picture contains all *macroblocks* of the *picture*. The only *pictures* that have a normative effect on the *decoding process* are primary coded pictures. See also *redundant coded picture*.
- **profile:** A specified subset of the syntax of this Recommendation | International Standard.
- **quantisation parameter:** A variable used by the *decoding process* for *scaling* of *transform coefficient levels*.
- **random access:** The act of starting the decoding process for a *bitstream* at a point other than the beginning of the stream.
- **raster scan:** A mapping of a rectangular two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the first top row of the two-dimensional pattern scanned from left to right, followed similarly by the second, third, etc. rows of the pattern (going down) each scanned from left to right.
- **raw byte sequence payload (RBSP):** A syntax structure containing an integer number of bytes that is encapsulated in a *NAL unit*. An RBSP is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and followed by zero or more subsequent bits equal to 0.
- **raw byte sequence payload (RBSP) stop bit:** A bit equal to 1 present within a *raw byte sequence payload (RBSP)* after a *string of data bits*. The location of the end of the *string of data bits* within an *RBSP* can be identified by searching from the end of the *RBSP* for the *RBSP stop bit*, which is the last non-zero bit in the *RBSP*.
- **recovery point:** A point in the *bitstream* at which the recovery of an exact or an approximate representation of the *decoded pictures* represented by the *bitstream* is achieved after a *random access* or *broken link*.

- **redundant coded picture:** A coded representation of a *picture* or a part of a *picture*. The content of a redundant coded picture shall not be used by the *decoding process* for a *bitstream* conforming to this Recommendation | International Standard. A *redundant coded picture* is not required to contain all *macroblocks* in the *primary coded picture*. Redundant coded pictures have no normative effect on the *decoding process*. See also *primary coded picture*.
- **reference field:** A *reference field* may be used for *inter prediction* when *P*, *SP*, and *B slices* of a *coded field* or *field macroblocks* of a *coded frame* are decoded. See also *reference picture*.
- **reference frame:** A *reference frame* may be used for *inter prediction* when *P*, *SP*, and *B slices* of a *coded frame* are decoded. See also *reference picture*.
- **reference index:** An index into a *reference picture list*.
- **reference picture:** A *picture* with `nal_ref_idc` not equal to 0. A *reference picture* contains samples that may be used for *inter prediction* in the *decoding process* of subsequent *pictures* in *decoding order*.
- **reference picture list:** A list of short-term *picture* numbers and long-term *picture* numbers that are assigned to *reference pictures*.
- **reference picture list 0:** A *reference picture list* used for *inter prediction* of a *P*, *B*, or *SP slice*. All *inter prediction* used for *P* and *SP slices* uses reference picture list 0. Reference picture list 0 is one of two *reference picture lists* used for *inter prediction* for a *B slice*, with the other being *reference picture list 1*.
- **reference picture list 1:** A *reference picture list* used for *inter prediction* of a *B slice*. Reference picture list 1 is one of two lists of *reference picture lists* used for *inter prediction* for a *B slice*, with the other being *reference picture list 0*.
- **reference picture marking:** Specifies, in the *bitstream*, how the *decoded pictures* are marked for *inter prediction*.
- **reserved:** The term reserved, when used in the clauses specifying some values of a particular *syntax element*, are for future use by ITU-T | ISO/IEC. These values shall not be used in *bitstreams* conforming to this Recommendation | International Standard, but may be used in future extensions of this Recommendation | International Standard by ITU-T | ISO/IEC.
- **residual:** The decoded difference between a *prediction* of a sample or data element and its decoded value.
- **run:** A number of consecutive data elements represented in the decoding process. In one context, the number of zero-valued *transform coefficient levels* preceding a non-zero *transform coefficient level* in the list of *transform coefficient levels* generated by a *zig-zag scan* or a *field scan*. In other contexts, run refers to a number of *macroblocks*.
- **sample aspect ratio:** Specifies, for assisting the display process, which is not specified in this Recommendation | International Standard, the ratio between the intended horizontal distance between the columns and the intended vertical distance between the rows of the *luma* sample array in a *frame*. Sample aspect ratio is expressed as  $h:v$ , where  $h$  is horizontal width and  $v$  is vertical height (in arbitrary units of spatial distance).

- **scaling:** The process of multiplying *transform coefficient levels* by a factor, resulting in *transform coefficients*.
- **SI slice:** A *slice* that is coded using *prediction* only from decoded samples within the same *slice* and using quantisation of the *prediction* samples. An SI slice can be coded such that its decoded samples can be constructed identically to an *SP slice*.
- **skipped macroblock:** A *macroblock* for which no data is coded other than an indication that the *macroblock* is to be decoded as "skipped". This indication may be common to several *macroblocks*.
- **slice:** An integer number of *macroblocks* or *macroblock pairs* ordered consecutively in the *raster scan* within a particular *slice group*. For the *primary coded picture*, the division of each *slice group* into slices is a *partitioning*. Although a slice contains *macroblocks* or *macroblock pairs* that are consecutive in the raster scan within a slice group, these *macroblocks* or *macroblock pairs* are not necessarily consecutive in the raster scan within the *picture*. The addresses of the *macroblocks* are derived from the address of the first *macroblock* in a slice (as represented in the *slice header*) and the *macroblock to slice group map*.
- **slice data partitioning:** A method of *partitioning* selected *syntax elements* into *syntax structures* based on a *category* associated with each *syntax element*.
- **slice group:** A subset of the *macroblocks* or *macroblock pairs* of a *picture*. The division of the *picture* into slice groups is a *partitioning* of the *picture*. The partitioning is specified by the *macroblock to slice group map*.
- **slice group map units:** The units of the *map unit to slice group map*.
- **slice header:** A part of a *coded slice* containing the data elements pertaining to the first or all *macroblocks* represented in the slice.
- **source:** Term used to describe the video material or some of its attributes before encoding.
- **SP slice:** A *slice* that is coded using *inter prediction* from previously-decoded *reference pictures*, using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*. An SP slice can be coded such that its decoded samples can be constructed identically to another SP slice or an *SI slice*.
- **start code prefix:** A unique sequence of three bytes equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*. The location of a *start code prefix* can be used by a decoder to identify the beginning of a new *NAL unit* and the end of a previous *NAL unit*. Emulation of *start code prefixes* is prevented within *NAL units* by the inclusion of *emulation prevention bytes*.
- **string of data bits (SODB):** A sequence of some number of bits representing *syntax elements* present within a *raw byte sequence payload* prior to the *raw byte sequence payload stop bit*. Within an *SODB*, the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit.
- **sub-macroblock:** One quarter of the samples of a *macroblock*, i.e., an 8x8 luma block and two 4x4 chroma blocks of which one corner is located at a corner of the *macroblock*.

- **sub-macroblock partition:** A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *sub-macroblock* for *inter prediction*.
  - **switching I slice:** See SI slice.
  - **switching P slice:** See SP slice.
  - **syntax element:** An element of data represented in the *bitstream*.
  - **syntax structure:** Zero or more *syntax elements* present together in the *bitstream* in a specified order.
  - **top field:** One of two *fields* that comprise a *frame*. Each row of a *top field* is spatially located immediately above the corresponding row of the *bottom field*.
  - **top macroblock (of a macroblock pair):** The *macroblock* within a *macroblock pair* that contains the samples in the top row of samples for the *macroblock pair*. For a *field macroblock pair*, the top macroblock represents the samples from the region of the *top field* of the *frame* that lie within the spatial region of the *macroblock pair*. For a *frame macroblock pair*, the top macroblock represents the samples of the *frame* that lie within the top half of the spatial region of the *macroblock pair*.
  - **transform coefficient:** A scalar quantity, considered to be in a frequency domain, that is associated with a particular one-dimensional or two-dimensional *frequency index* in an *inverse transform* part of the *decoding process*.
  - **transform coefficient level:** An integer quantity representing the value associated with a particular two-dimensional frequency index in the *decoding process* prior to *scaling* for computation of a *transform coefficient* value.
  - **universal unique identifier (UUID):** An identifier that is unique with respect to the space of all universal unique identifiers.
  - **unspecified:** The term unspecified, when used in the clauses specifying some values of a particular *syntax element*, indicates that the values have no specified meaning in this Recommendation | International Standard and will not have a specified meaning in the future as an integral part of this Recommendation | International Standard.
  - **variable length coding (VLC):** A reversible procedure for entropy coding that assigns shorter bit strings to *symbols* expected to be more frequent and longer bit strings to *symbols* expected to be less frequent.
  - **zig-zag scan:** A specific sequential ordering of *transform coefficient levels* from (approximately) the lowest spatial frequency to the highest. Zig-zag scan is used for *transform coefficient levels* in *frame macroblocks*.
- 
- **CATV** Cable TV on optical networks, copper, etc.
  - **DBS** Direct broadcast satellite video services

- DSL     Digital subscriber line video services
- DTTB   Digital terrestrial television broadcasting
- ISM     Interactive storage media (optical disks, etc.)
- MMM   Multimedia mailing
- MSPN   Multimedia services over packet networks
- RTC     Real-time conversational services (videoconferencing, videophone, etc.)
- RVS     Remote video surveillance
- SSM     Serial storage media (digital VTR, etc.)



## 附录一 CAVLC 相关码表

表 1 coeff\_token 到 TotalCoeff( coeff\_token ) 和 TrailingOnes( coeff\_token )映射

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	-
0	7	0000 0000 0101 1	0000 0001 111	0001 000	0110 00	-
1	7	0000 0000 0111 0	0000 0011 0	0010 10	0110 01	-
2	7	0000 0000 101	0000 0010 1	0010 01	0110 10	-

3	7	0000 0010 0	0001 00	1000	0110 11	-
0	8	0000 0000 0100 0	0000 0001 011	0000 1111	0111 00	-
1	8	0000 0000 0101 0	0000 0001 110	0001 110	0111 01	-
2	8	0000 0000 0110 1	0000 0001 101	0001 101	0111 10	-
3	8	0000 0001 00	0000 100	0110 1	0111 11	-
0	9	0000 0000 0011 11	0000 0000 1111	0000 1011	1000 00	-
1	9	0000 0000 0011 10	0000 0001 010	0000 1110	1000 01	-
2	9	0000 0000 0100 1	0000 0001 001	0001 010	1000 10	-
3	9	0000 0000 100	0000 0010 0	0011 00	1000 11	-
0	10	0000 0000 0010 11	0000 0000 1011	0000 0111 1	1001 00	-
1	10	0000 0000 0010 10	0000 0000 1110	0000 1010	1001 01	-
2	10	0000 0000 0011 01	0000 0000 1101	0000 1101	1001 10	-
3	10	0000 0000 0110 0	0000 0001 100	0001 100	1001 11	-
0	11	0000 0000 0001 111	0000 0000 1000	0000 0101 1	1010 00	-
1	11	0000 0000 0001 110	0000 0000 1010	0000 0111 0	1010 01	-
2	11	0000 0000 0010 01	0000 0000 1001	0000 1001	1010 10	-
3	11	0000 0000 0011 00	0000 0001 000	0000 1100	1010 11	-
0	12	0000 0000 0001 011	0000 0000 0111 1	0000 0100 0	1011 00	-
1	12	0000 0000 0001 010	0000 0000 0111 0	0000 0101 0	1011 01	-
2	12	0000 0000 0001 101	0000 0000 0110 1	0000 0110 1	1011 10	-
3	12	0000 0000 0010 00	0000 0000 1100	0000 1000	1011 11	-
0	13	0000 0000 0000 1111	0000 0000 0101 1	0000 0011 01	1100 00	-
1	13	0000 0000 0000 001	0000 0000 0101 0	0000 0011 1	1100 01	-

2	13	0000 0000 0001 001	0000 0000 0100 1	0000 0100 1	1100 10	-
3	13	0000 0000 0001 100	0000 0000 0110 0	0000 0110 0	1100 11	-
0	14	0000 0000 0000 1011	0000 0000 0011 1	0000 0010 01	1101 00	-
1	14	0000 0000 0000 1110	0000 0000 0010 11	0000 0011 00	1101 01	-
2	14	0000 0000 0000 1101	0000 0000 0011 0	0000 0010 11	1101 10	-
3	14	0000 0000 0001 000	0000 0000 0100 0	0000 0010 10	1101 11	-
0	15	0000 0000 0000 0111	0000 0000 0010 01	0000 0001 01	1110 00	-
1	15	0000 0000 0000 1010	0000 0000 0010 00	0000 0010 00	1110 01	-
2	15	0000 0000 0000 1001	0000 0000 0010 10	0000 0001 11	1110 10	-
3	15	0000 0000 0000 1100	0000 0000 0000 1	0000 0001 10	1110 11	-
0	16	0000 0000 0000 0100	0000 0000 0001 11	0000 0000 01	1111 00	-
1	16	0000 0000 0000 0110	0000 0000 0001 10	0000 0001 00	1111 01	-
2	16	0000 0000 0000 0101	0000 0000 0001 01	0000 0000 11	1111 10	-
3	16	0000 0000 0000 1000	0000 0000 0001 00	0000 0000 10	1111 11	-

表 2 level\_prefix 码表

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

表 3 TotalCoeff( coeff\_token ) 为 1~7 时 4x4 块的 total\_zeros 表

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						

表 4 TotalCoeff( coeff\_token ) 为 8~15 时 4x4 块的 total\_zeros 表

total_zeros	TotalCoeff( coeff_token )							
	8	9	10	11	12	13	14	15
0	0000 01	0000 01	0000 1	0000	0000	000	00	0
1	0001	0000 00	0000 0	0001	0001	001	01	1
2	0000 1	0001	001	001	01	1	1	
3	011	11	11	010	1	01		
4	11	10	10	1	001			
5	10	001	01	011				
6	010	01	0001					
7	001	0000 1						
8	0000 00							

表 5 色度 DC 2x2 块的 total\_zeros 表

total_zeros	TotalCoeff( coeff_token )		
	1	2	3
0	1	1	1
1	01	01	0
2	001	00	
3	000		

表 6 run\_before 表

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

附录二 CABAC 相关码表

表 1 ctxIdx 为 0~398 时变量 m 和 n 的取值表

Initialisation variables	ctxIdx										
	0	1	2	3	4	5	6	7	8	9	10
m	20	2	3	20	2	3	-28	-23	-6	-1	7
n	-15	54	74	-15	54	74	127	104	53	54	51

Value of cabac_init_idc	Initialisation variables	ctxIdx												
		11	12	13	14	15	16	17	18	19	20	21	22	23
0	m	23	23	21	1	0	-37	5	-13	-11	1	12	-4	17
	n	33	2	0	9	49	118	57	78	65	62	49	73	50
1	m	22	34	16	-2	4	-29	2	-6	-13	5	9	-3	10
	n	25	0	0	9	41	118	65	71	79	52	50	70	54
2	m	29	25	14	-10	-3	-27	26	-4	-24	5	6	-17	14
	n	16	0	0	51	62	99	16	85	102	57	57	73	57

Value of cabac_init_ idc	Initialisat ion variables	ctxIdx																
		24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	
0	m	18	9	29	26	16	9	-4 6	-2 0	1	-1 3	-1 1	1	-6	-1 7	-6	9	
	n	64	43	0	67	90	10 4	12 7	10 4	67	78	65	62	86	95	61	45	
1	m	26	19	40	57	41	26	-4 5	-1 5	-4	-6	-1 3	5	6	-1 3	0	8	
	n	34	22	0	2	36	69	12 7	10 1	76	71	79	52	69	90	52	43	
2	m	20	20	29	54	37	12	-3 2	-2 2	-2	-4	-2 4	5	-6	-1 4	-6	4	
	n	40	10	0	0	42	97	12 7	11 7	74	85	10 2	57	93	88	44	55	



Value of cabac_init_id c	Initialisation n variables	ctxIdx													
		40	41	42	43	44	45	46	47	48	49	50	51	52	53
0	m	-3	-6	-11	6	7	-5	2	0	-3	-10	5	4	-3	0
	n	69	81	96	55	67	86	88	58	76	94	54	69	81	88
1	m	-2	-5	-10	2	2	-3	-3	1	-3	-6	0	-3	-7	-5
	n	69	82	96	59	75	87	100	56	74	85	59	81	86	95
2	m	-11	-15	-21	19	20	4	6	1	-5	-13	5	6	-3	-1
	n	89	103	116	57	58	84	96	63	85	106	63	75	90	101

Value of cabac_init_idc	Initialisation variables	ctxIdx					
		54	55	56	57	58	59
0	m	-7	-5	-4	-5	-7	1
	n	67	74	74	80	72	58
1	m	-1	-1	1	-2	-5	0
	n	66	77	70	86	72	61
2	m	3	-4	-2	-12	-7	1
	n	55	79	75	97	50	60

Initialisation variables	ctxIdx									
	60	61	62	63	64	65	66	67	68	69
m	0	0	0	0	-9	4	0	-7	13	3
n	41	63	63	63	83	86	97	72	41	62



ctxI dx	I and SI slices		Value of cabac_init_idc						ctxId x	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
105	-7	93	-2	85	$-\frac{1}{3}$	$\frac{10}{3}$	-4	86	136	-13	$\frac{10}{1}$	5	53	0	58	-5	75
106	$-\frac{1}{1}$	87	-6	78	$-\frac{1}{3}$	91	$-\frac{1}{2}$	88	137	-13	91	-2	61	-1	60	-8	80
107	-3	77	-1	75	-9	89	-5	82	138	-12	94	0	56	-3	61	-21	83
108	-5	71	-7	77	$-\frac{1}{4}$	92	-3	72	139	-10	88	0	56	-8	67	-21	64
109	-4	63	2	54	-8	76	-4	67	140	-16	84	-13	63	-25	84	-13	31
110	-4	68	5	50	$-\frac{1}{2}$	87	-8	72	141	-10	86	-5	60	-14	74	-25	64
111	$-\frac{1}{2}$	84	-3	68	$-\frac{2}{3}$	$\frac{11}{0}$	$-\frac{1}{6}$	89	142	-7	83	-1	62	-5	65	-29	94
112	-7	62	1	50	$-\frac{2}{4}$	$\frac{10}{5}$	-9	69	143	-13	87	4	57	5	52	9	75
113	-7	65	6	42	$-\frac{1}{0}$	78	-1	59	144	-19	94	-6	69	2	57	17	63
114	8	61	-4	81	$-\frac{2}{0}$	$\frac{11}{2}$	5	66	145	1	70	4	57	0	61	-8	74
115	5	56	1	63	$-\frac{1}{7}$	99	4	57	146	0	72	14	39	-9	69	-5	35
116	-2	66	-4	70	$-\frac{7}{8}$	$\frac{12}{7}$	-4	71	147	-5	74	4	51	-11	70	-2	27
117	1	64	0	67	$-\frac{7}{0}$	$\frac{12}{7}$	-2	71	148	18	59	13	68	18	55	13	91
118	0	61	2	57	$-\frac{5}{0}$	$\frac{12}{7}$	2	58	149	-8	$\frac{10}{2}$	3	64	-4	71	3	65
119	-2	78	-2	76	$-\frac{4}{6}$	$\frac{12}{7}$	-1	74	150	-15	$\frac{10}{0}$	1	61	0	58	-7	69
120	1	50	11	35	-4	66	-4	44	151	0	95	9	63	7	61	8	77
121	7	52	4	64	-5	78	-1	69	152	-4	75	7	50	9	41	-10	66
122	10	35	1	61	-4	71	0	62	153	2	72	16	39	18	25	3	62
123	0	44	11	35	-8	72	-7	51	154	-11	75	5	44	9	32	-3	68
124	11	38	18	25	2	59	-4	47	155	-3	71	4	52	5	43	-20	81
125	1	45	12	24	-1	55	-6	42	156	15	46	11	48	9	47	0	30
126	0	46	13	29	-7	70	-3	41	157	-13	69	-5	60	0	44	1	7
127	5	44	13	36	-6	75	-6	53	158	0	62	-1	59	0	51	-3	23
128	31	17	$-\frac{1}{0}$	93	-8	89	8	76	159	0	65	0	59	2	46	-21	74

<b>129</b>	1	51	-7	73	$-\frac{3}{4}$	$\frac{11}{9}$	-9	78	<b>160</b>	21	37	22	33	19	38	16	66
<b>130</b>	7	50	-2	73	-3	75	$-\frac{1}{1}$	83	<b>161</b>	-15	72	5	44	-4	66	-23	$\frac{12}{4}$
<b>131</b>	28	19	13	46	32	20	9	52	<b>162</b>	9	57	14	43	15	38	17	37
<b>132</b>	16	33	9	49	30	22	0	67	<b>163</b>	16	54	-1	78	12	42	44	-18
<b>133</b>	14	62	-7	$\frac{10}{0}$	$-\frac{4}{4}$	$\frac{12}{7}$	-5	90	<b>164</b>	0	62	0	60	9	34	50	-34
<b>134</b>	$-\frac{1}{3}$	$\frac{10}{8}$	9	53	0	54	1	67	<b>165</b>	12	72	9	69	0	89	-22	$\frac{12}{7}$
<b>135</b>	$-\frac{1}{5}$	$\frac{10}{0}$	2	53	-5	61	$-\frac{1}{5}$	72									

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
166	24	0	11	28	4	45	4	39	197	26	-17	28	3	36	-28	28	-3
167	15	9	2	40	10	28	0	42	198	30	-25	28	4	38	-28	24	10
168	8	25	3	44	10	31	7	34	199	28	-20	32	0	38	-27	27	0
169	13	18	0	49	33	-11	11	29	200	33	-23	34	-1	34	-18	34	-14
170	15	9	0	46	52	-43	8	31	201	37	-27	30	6	35	-16	52	-44
171	13	19	2	44	18	15	6	37	202	33	-23	30	6	34	-14	39	-24
172	10	37	2	51	28	0	7	42	203	40	-28	32	9	32	-8	19	17
173	12	18	0	47	35	-22	3	40	204	38	-17	31	19	37	-6	31	25
174	6	29	4	39	38	-25	8	33	205	33	-11	26	27	35	0	36	29
175	20	33	2	62	34	0	13	43	206	40	-15	26	30	30	10	24	33
176	15	30	6	46	39	-18	13	36	207	41	-6	37	20	28	18	34	15
177	4	45	0	54	32	-12	4	47	208	38	1	28	34	26	25	30	20
178	1	58	3	54	102	-94	3	55	209	41	17	17	70	29	41	22	73
179	0	62	2	58	0	0	2	58	210	30	-6	1	67	0	75	20	34
180	7	61	4	63	56	-15	6	60	211	27	3	5	59	2	72	19	31
181	12	38	6	51	33	-4	8	44	212	26	22	9	67	8	77	27	44
182	11	45	6	57	29	10	11	44	213	37	-16	16	30	14	35	19	16
183	15	39	7	53	37	-5	14	42	214	35	-4	18	32	18	31	15	36
184	11	42	6	52	51	-29	7	48	215	38	-8	18	35	17	35	15	36
185	13	44	6	55	39	-9	4	56	216	38	-3	22	29	21	30	21	28
186	16	45	11	45	52	-34	4	52	217	37	3	24	31	17	45	25	21
187	12	41	14	36	69	-58	13	37	218	38	5	23	38	20	42	30	20



ctxId x	I and SI slices		Value of cabac_init_idc						ctxId x	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n		
227	-3	7 1	-6	76	-2 3	11 2	-2 4	11 5	252	-1 2	73	-6	5 5	-1 6	7 2	-1 4	75
228	-6	4 2	-2	44	-1 5	71	-2 2	82	253	-8	76	0	5 8	-7	6 9	-1 0	79
229	-5	5 0	0	45	-7	61	-9	62	254	-7	80	0	6 4	-4	6 9	-9	83
230	-3	5 4	0	52	0	53	0	53	255	-9	88	-3	7 4	-5	7 4	-1 2	92
231	-2	6 2	-3	64	-5	66	0	59	256	-1 7	11 0	-1 0	9 0	-9	8 6	-1 8	10 8
232	0	5 8	-2	59	-1 1	77	-1 4	85	257	-1 1	97	0	7 0	2	6 6	-4	79
233	1	6 3	-4	70	-9	80	-1 3	89	258	-2 0	84	-4	2 9	-9	3 4	-2 2	69
234	-2	7 2	-4	75	-9	84	-1 3	94	259	-1 1	79	5	3 1	1	3 2	-1 6	75
235	-1	7 4	-8	82	-1 0	87	-1 1	92	260	-6	73	7	4 2	11	3 1	-2	58
236	-9	9 1	-1 7	10 2	-3 4	12 7	-2 9	12 7	261	-4	74	1	5 9	5	5 2	1	58
237	-5	6 7	-9	77	-2 1	10 1	-2 1	10 0	262	-1 3	86	-2	5 8	-2	5 5	-1 3	78
238	-5	2 7	3	24	-3	39	-1 4	57	263	-1 3	96	-3	7 2	-2	6 7	-9	83
239	-3	3 9	0	42	-5	53	-1 2	67	264	-1 1	97	-3	8 1	0	7 3	-4	81
240	-2	4 4	0	48	-7	61	-1 1	71	265	-1 9	11 7	-1 1	9 7	-8	8 9	-1 3	99
241	0	4 6	0	55	-1 1	75	-1 0	77	266	-8	78	0	5 8	3	5 2	-1 3	81
242	-1 6	6 4	-6	59	-1 5	77	-2 1	85	267	-5	33	8	5	7	4	-6	38
243	-8	6 8	-7	71	-1 7	91	-1 6	88	268	-4	48	10	1 4	10	8	-1 3	62
244	-1 0	7 8	-1 2	83	-2 5	10 7	-2 3	10 4	269	-2	53	14	1 8	17	8	-6	58
245	-6	7 7	-1 1	87	-2 5	11 1	-1 5	98	270	-3	62	13	2 7	16	1 9	-2	59
246	-1 0	8 6	-3 0	11 9	-2 8	12 2	-3 7	12 7	271	-1 3	71	2	4 0	3	3 7	-1 6	73
247	-1 2	9 2	1	58	-1 1	76	-1 0	82	272	-1 0	79	0	5 8	-1	6 1	-1 0	76

<b>248</b>	$-\frac{1}{5}$	$\frac{5}{5}$	-3	29	$-\frac{1}{0}$	44	-8	48	<b>273</b>	$-\frac{1}{2}$	86	-3	$\frac{7}{0}$	-5	$\frac{7}{3}$	$-\frac{1}{3}$	86
<b>249</b>	$-\frac{1}{0}$	$\frac{6}{0}$	-1	36	$-\frac{1}{0}$	52	-8	61	<b>274</b>	$-\frac{1}{3}$	90	-6	$\frac{7}{9}$	-1	$\frac{7}{0}$	-9	83
<b>250</b>	-6	$\frac{6}{2}$	1	38	$-\frac{1}{0}$	57	-8	66	<b>275</b>	$-\frac{1}{4}$	97	-8	$\frac{8}{5}$	-4	$\frac{7}{8}$	$-\frac{1}{0}$	87
<b>251</b>	-4	$\frac{6}{5}$	2	43	-9	58	-7	70									

ctxId x	I and SI slices		Value of cabac_init_idc						ctxId x	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n		
277	-6	93	-13	10 6	-2 1	12 6	-2 2	12 7	308	-1 6	9 6	-1	5 1	-1 6	7 7	-1 0	6 7
278	-6	84	-16	10 6	-2 3	12 4	-2 5	12 7	309	-7	8 8	7	4 9	-2	6 4	1	6 8
279	-8	79	-10	87	-2 0	11 0	-2 5	12 0	310	-8	8 5	8	5 2	2	6 1	0	7 7
280	0	66	-21	11 4	-2 6	12 6	-2 7	12 7	311	-7	8 5	9	4 1	-6	6 7	2	6 4
281	-1	71	-18	11 0	-2 5	12 4	-1 9	11 4	312	-9	8 5	6	4 7	-3	6 4	0	6 8
282	0	62	-14	98	-1 7	10 5	-2 3	11 7	313	-1 3	8 8	2	5 5	2	5 7	-5	7 8
283	-2	60	-22	11 0	-2 7	12 1	-2 5	11 8	314	4	6 6	1 3	4 1	-3	6 5	7	5 5
284	-2	59	-21	10 6	-2 7	11 7	-2 6	11 7	315	-3	7 7	1 0	4 4	-3	6 6	5	5 9
285	-5	75	-18	10 3	-1 7	10 2	-2 4	11 3	316	-3	7 6	6	5 0	0	6 2	2	6 5
286	-3	62	-21	10 7	-2 6	11 7	-2 8	11 8	317	-6	7 6	5	5 3	9	5 1	14	5 4
287	-4	58	-23	10 8	-2 7	11 6	-3 1	12 0	318	10	5 8	1 3	4 9	-1	6 6	15	4 4
288	-9	66	-26	11 2	-3 3	12 2	-3 7	12 4	319	-1	7 6	4	6 3	-2	7 1	5	6 0
289	-1	79	-10	96	-1 0	95	-1 0	94	320	-1	8 3	6	6 4	-2	7 5	2	7 0
290	0	71	-12	95	-1 4	10 0	-1 5	10 2	321	-7	9 9	-2	6 9	-1	7 0	-2	7 6
291	3	68	-5	91	-8	95	-1 0	99	322	-1 4	9 5	-2	5 9	-9	7 2	-1 8	8 6
292	10	44	-9	93	-1 7	11 1	-1 3	10 6	323	2	9 5	6	7 0	14	6 0	12	7 0
293	-7	62	-22	94	-2 8	11 4	-5 0	12 7	324	0	7 6	1 0	4 4	16	3 7	5	6 4





ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n		
338	15	6	14	11	19	-6	17	-13	369	32	-26	31	-4	40	-37	37	-17
339	6	19	11	14	18	-6	16	-9	370	37	-30	27	6	38	-30	32	1
340	7	16	9	11	14	0	17	-12	371	44	-32	34	8	46	-33	34	15
341	12	14	18	11	26	-12	27	-21	372	34	-18	30	10	42	-30	29	15
342	18	13	21	9	31	-16	37	-30	373	34	-15	24	22	40	-24	24	25
343	13	11	23	-2	33	-25	41	-40	374	40	-15	33	19	49	-29	34	22
344	13	15	32	-15	33	-22	42	-41	375	33	-7	22	32	38	-12	31	16
345	15	16	32	-15	37	-28	48	-47	376	35	-5	26	31	40	-10	35	18
346	12	23	34	-21	39	-30	39	-32	377	33	0	21	41	38	-3	31	28
347	13	23	39	-23	42	-30	46	-40	378	38	2	26	44	46	-5	33	41
348	15	20	42	-33	47	-42	52	-51	379	33	13	23	47	31	20	36	28
349	14	26	41	-31	45	-36	46	-41	380	23	35	16	65	29	30	27	47
350	14	44	46	-28	49	-34	52	-39	381	13	58	14	71	25	44	21	62
351	17	40	38	-12	41	-17	43	-19	382	29	-3	8	60	12	48	18	31
352	17	47	21	29	32	9	32	11	383	26	0	6	63	11	49	19	26
353	24	17	45	-24	69	-71	61	-55	384	22	30	17	65	26	45	36	24
354	21	21	53	-45	63	-63	56	-46	385	31	-7	21	24	22	22	24	23
355	25	22	48	-26	66	-64	62	-50	386	35	-15	23	20	23	22	27	16
356	31	27	65	-43	77	-74	81	-67	387	34	-3	26	23	27	21	24	30
357	22	29	43	-19	54	-39	45	-20	388	34	3	27	32	33	20	31	29
358	19	35	39	-10	52	-35	35	-2	389	36	-1	28	23	26	28	22	41
359	14	50	30	9	41	-10	28	15	390	34	5	28	24	30	24	22	42
360	10	57	18	26	36	0	34	1	391	32	11	23	40	27	34	16	60
361	7	63	20	27	40	-1	39	1	392	35	5	24	32	18	42	15	52
362	-2	77	0	57	30	14	30	17	393	34	12	28	29	25	39	14	60
363	-4	82	-14	82	28	26	20	38	394	39	11	23	42	18	50	3	78
364	-3	94	-5	75	23	37	18	45	395	30	29	19	57	12	70	-16	123
365	9	69	-19	97	12	55	15	54	396	34	26	22	53	21	54	21	53
366	-12	109	-35	125	11	65	0	79	397	29	39	22	61	14	71	22	56
367	36	-35	27	0	37	-33	36	-16	398	19	66	11	86	11	83	25	61
368	36	-34	28	0	39	-36	37	-14									

表 2 I 片中宏块类型二进制流

mb_type	Bin string						
0 (I_4x4)	0						
1 (I_16x16_0_0_0)	1	0	0	0	0	0	
2 (I_16x16_1_0_0)	1	0	0	0	0	1	
3 (I_16x16_2_0_0)	1	0	0	0	1	0	
4 (I_16x16_3_0_0)	1	0	0	0	1	1	
5 (I_16x16_0_1_0)	1	0	0	1	0	0	0
6 (I_16x16_1_1_0)	1	0	0	1	0	0	1
7 (I_16x16_2_1_0)	1	0	0	1	0	1	0
8 (I_16x16_3_1_0)	1	0	0	1	0	1	1
9 (I_16x16_0_2_0)	1	0	0	1	1	0	0
10 (I_16x16_1_2_0)	1	0	0	1	1	0	1
11 (I_16x16_2_2_0)	1	0	0	1	1	1	0
12 (I_16x16_3_2_0)	1	0	0	1	1	1	1
13 (I_16x16_0_0_1)	1	0	1	0	0	0	
14 (I_16x16_1_0_1)	1	0	1	0	0	1	
15 (I_16x16_2_0_1)	1	0	1	0	1	0	
16 (I_16x16_3_0_1)	1	0	1	0	1	1	
17 (I_16x16_0_1_1)	1	0	1	1	0	0	0
18 (I_16x16_1_1_1)	1	0	1	1	0	0	1
19 (I_16x16_2_1_1)	1	0	1	1	0	1	0
20 (I_16x16_3_1_1)	1	0	1	1	0	1	1
21 (I_16x16_0_2_1)	1	0	1	1	1	0	0
22 (I_16x16_1_2_1)	1	0	1	1	1	0	1
23 (I_16x16_2_2_1)	1	0	1	1	1	1	0
24 (I_16x16_3_2_1)	1	0	1	1	1	1	1
25 (I_PCM)	1	1					
binIdx	0	1	2	3	4	5	6

表 3 IP, SP, 和 B 片中宏块类型二进制流

Slice type	Value (name) of mb_type	Bin string						
P, SP slice	0 (P_L0_16x16)	0	0	0				
	1 (P_L0_L0_16x8)	0	1	1				
	2 (P_L0_L0_8x16)	0	1	0				
	3 (P_8x8)	0	0	1				
	4 (P_8x8ref0)	na						
	5 to 30 (Intra, prefix only)	1						
B slice	0 (B_Direct_16x16)	0						
	1 (B_L0_16x16)	1	0	0				
	2 (B_L1_16x16)	1	0	1				
	3 (B_Bi_16x16)	1	1	0	0	0	0	
	4 (B_L0_L0_16x8)	1	1	0	0	0	1	
	5 (B_L0_L0_8x16)	1	1	0	0	1	0	
	6 (B_L1_L1_16x8)	1	1	0	0	1	1	
	7 (B_L1_L1_8x16)	1	1	0	1	0	0	
	8 (B_L0_L1_16x8)	1	1	0	1	0	1	
	9 (B_L0_L1_8x16)	1	1	0	1	1	0	
	10 (B_L1_L0_16x8)	1	1	0	1	1	1	
	11 (B_L1_L0_8x16)	1	1	1	1	1	0	
	12 (B_L0_Bi_16x8)	1	1	1	0	0	0	0
	13 (B_L0_Bi_8x16)	1	1	1	0	0	0	1
	14 (B_L1_Bi_16x8)	1	1	1	0	0	1	0
	15 (B_L1_Bi_8x16)	1	1	1	0	0	1	1
	16 (B_Bi_L0_16x8)	1	1	1	0	1	0	0
	17 (B_Bi_L0_8x16)	1	1	1	0	1	0	1
	18 (B_Bi_L1_16x8)	1	1	1	0	1	1	0
	19 (B_Bi_L1_8x16)	1	1	1	0	1	1	1
	20 (B_Bi_Bi_16x8)	1	1	1	1	0	0	0
	21 (B_Bi_Bi_8x16)	1	1	1	1	0	0	1
	22 (B_8x8)	1	1	1	1	1	1	
	23 to 48 (Intra, prefix only)	1	1	1	1	0	1	
binIdx		0	1	2	3	4	5	6

表 4 IP, SP, 和 B 片中子宏块类型二进制流

Slice type	Value (name) of sub_mb_type	Bin string					
P, SP slice	0 (P_L0_8x8)	1					
	1 (P_L0_8x4)	0	0				
	2 (P_L0_4x8)	0	1	1			
	3 (P_L0_4x4)	0	1	0			
B slice	0 (B_Direct_8x8)	0					
	1 (B_L0_8x8)	1	0	0			
	2 (B_L1_8x8)	1	0	1			
	3 (B_Bi_8x8)	1	1	0	0	0	
	4 (B_L0_8x4)	1	1	0	0	1	
	5 (B_L0_4x8)	1	1	0	1	0	
	6 (B_L1_8x4)	1	1	0	1	1	
	7 (B_L1_4x8)	1	1	1	0	0	0
	8 (B_Bi_8x4)	1	1	1	0	0	1
	9 (B_Bi_4x8)	1	1	1	0	1	0
	10 (B_L0_4x4)	1	1	1	0	1	1
	11 (B_L1_4x4)	1	1	1	1	0	
	12 (B_Bi_4x4)	1	1	1	1	1	
binIdx		0	1	2	3	4	5

表 5 TabLPS 取值表

pStateIdx	qCodIRangeIdx				pStateIdx	qCodIRangeIdx			
	0	1	2	3		0	1	2	3
0	128	176	208	240	32	27	33	39	45
1	128	167	197	227	33	26	31	37	43
2	128	158	187	216	34	24	30	35	41
3	123	150	178	205	35	23	28	33	39
4	116	142	169	195	36	22	27	32	37
5	111	135	160	185	37	21	26	30	35
6	105	128	152	175	38	20	24	29	33
7	100	122	144	166	39	19	23	27	31
8	95	116	137	158	40	18	22	26	30
9	90	110	130	150	41	17	21	25	28
10	85	104	123	142	42	16	20	23	27
11	81	99	117	135	43	15	19	22	25
12	77	94	111	128	44	14	18	21	24
13	73	89	105	122	45	14	17	20	23
14	69	85	100	116	46	13	16	19	22
15	66	80	95	110	47	12	15	18	21
16	62	76	90	104	48	12	14	17	20
17	59	72	86	99	49	11	14	16	19
18	56	69	81	94	50	11	13	15	18
19	53	65	77	89	51	10	12	15	17
20	51	62	73	85	52	10	12	14	16
21	48	59	69	80	53	9	11	13	15
22	46	56	66	76	54	9	11	12	14
23	43	53	63	72	55	8	10	12	14
24	41	50	59	69	56	8	9	11	13
25	39	48	56	65	57	7	9	11	12
26	37	45	54	62	58	7	9	10	12
27	35	43	51	59	59	7	8	10	11
28	33	41	48	56	60	6	8	9	11
29	32	39	46	53	61	6	7	9	10
30	30	37	43	50	62	6	7	8	9
31	29	35	41	48	63	2	2	2	2

表 6 State transition 表

<b>pStateIdx</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>
<b>transIdxLPS</b>	0	0	1	2	2	4	4	5	6	7	8	9	9	11	11	12
<b>transIdxMPS</b>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
<b>pStateIdx</b>	<b>16</b>	<b>17</b>	<b>18</b>	<b>19</b>	<b>20</b>	<b>21</b>	<b>22</b>	<b>23</b>	<b>24</b>	<b>25</b>	<b>26</b>	<b>27</b>	<b>28</b>	<b>29</b>	<b>30</b>	<b>31</b>
<b>transIdxLPS</b>	13	13	15	15	16	16	18	18	19	19	21	21	22	22	23	24
<b>transIdxMPS</b>	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
<b>pStateIdx</b>	<b>32</b>	<b>33</b>	<b>34</b>	<b>35</b>	<b>36</b>	<b>37</b>	<b>38</b>	<b>39</b>	<b>40</b>	<b>41</b>	<b>42</b>	<b>43</b>	<b>44</b>	<b>45</b>	<b>46</b>	<b>47</b>
<b>transIdxLPS</b>	24	25	26	26	27	27	28	29	29	30	30	30	31	32	32	33
<b>transIdxMPS</b>	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
<b>pStateIdx</b>	<b>48</b>	<b>49</b>	<b>50</b>	<b>51</b>	<b>52</b>	<b>53</b>	<b>54</b>	<b>55</b>	<b>56</b>	<b>57</b>	<b>58</b>	<b>59</b>	<b>60</b>	<b>61</b>	<b>62</b>	<b>63</b>
<b>transIdxLPS</b>	33	33	34	34	35	35	35	36	36	36	37	37	37	38	38	63
<b>transIdxMPS</b>	49	50	51	52	53	54	55	56	57	58	59	60	61	62	62	63

附录三 H. 264 档次和级

表 1 级限制

级数	最大宏块处理速率 MaxMBPS (MB/s)	最大帧尺寸 MaxFS (MBs)	最大解码缓冲 区尺寸 MaxDPB (1024 bytes)	最大视频比特率 MaxBR (1000 bits/s or 1200 bits/s)	最大 CPB 尺寸 MaxCPB (1000 bits or 1200 bits)	垂直 MV 范围 MaxVmvR (亮度帧样电)	最小压缩率 MinCR	每两个连续 MB 的 MV 最大数 MaxMvsPer2Mb
1	1 485	99	148.5	64	175	[-64,+63.75]	2	-
1.1	3 000	396	337.5	192	500	[-128,+127.75]	2	-
1.2	6 000	396	891.0	384	1 000	[-128,+127.75]	2	-
1.3	11 880	396	891.0	768	2 000	[-128,+127.75]	2	-
2	11 880	396	891.0	2 000	2 000	[-128,+127.75]	2	-
2.1	19 800	792	1 782.0	4 000	4 000	[-256,+255.75]	2	-
2.2	20 250	1 620	3 037.5	4 000	4 000	[-256,+255.75]	2	-
3	40 500	1 620	3 037.5	10 000	10 000	[-256,+255.75]	2	32
3.1	108 000	3 600	6 750.0	14 000	14 000	[-512,+511.75]	4	16
3.2	216 000	5 120	7 680.0	20 000	20 000	[-512,+511.75]	4	16
4	245 760	8 192	12 288.0	20 000	25 000	[-512,+511.75]	4	16
4.1	245 760	8 192	12 288.0	50 000	62 500	[-512,+511.75]	2	16
4.2	491 520	8 192	12 288.0	50 000	62 500	[-512,+511.75]	2	16
5	589 824	22 080	41 310.0	135 000	135 000	[-512,+511.75]	2	16
5.1	983 040	36 864	69 120.0	240 000	240 000	[-512,+511.75]	2	16

表 2 基本档次级限制

Level	number	MaxSubMbRectSize
1		576
1.1		576
1.2		576
1.3		576
2		576
2.1		576
2.2		576
3		576
3.1		-
3.2		-
4		-
4.1		-
4.2		-
5		-
5.1		-

表 3 主要档次级限制

Level number	SliceRate	MinLumaBiPredSize	direct_8x8_inference_flag	frame_mbs_only_flag
1	-	-	-	1
1.1	-	-	-	1
1.2	-	-	-	1
1.3	-	-	-	1
2	-	-	-	1
2.1	-	-	-	-
2.2	-	-	-	-
3	22	-	1	-
3.1	60	8x8	1	-
3.2	60	8x8	1	-
4	60	8x8	1	-
4.1	24	8x8	1	-
4.2	24	8x8	1	1
5	24	8x8	1	1
5.1	24	8x8	1	1

表 4 扩展档次级限制

Level number	MaxSubMbRectSize	MinLumaBiPredSize	frame_mbs_only_flag
1	576	-	1
1.1	576	-	1
1.2	576	-	1
1.3	576	-	1
2	576	-	1
2.1	576	-	-
2.2	576	-	-
3	576	-	-
3.1	-	8x8	-
3.2	-	8x8	-
4	-	8x8	-
4.1	-	8x8	-
4.2	-	8x8	1
5	-	8x8	1
5.1	-	8x8	1

表 5 最大帧率举例



级数:					1	1.1	1.2	1.3	2	2.1	2.2
最大帧尺寸(宏块):					99	396	396	396	396	792	1 620
最大宏块数/s:					1 485	3 000	6 000	11 880	11 880	19 800	20 250
最大帧尺寸(样点):					25 344	101 376	101 376	101 376	101 376	202 752	414 720
最大样点数/s:					380 160	768 000	1 536 000	3 041 280	3 041 280	5 068 800	5 184 000
格式	亮度 宽度	亮度 高度	总宏 块数	亮度样 点数							
SQCIF	128	96	48	12 288	30.9	62.5	125.0	172.0	172.0	172.0	172.0
QCIF	176	144	99	25 344	15.0	30.3	60.6	120.0	120.0	172.0	172.0
QVGA	320	240	300	76 800	-	10.0	20.0	39.6	39.6	66.0	67.5
525 SIF	352	240	330	84 480	-	9.1	18.2	36.0	36.0	60.0	61.4
CIF	352	288	396	101 376	-	7.6	15.2	30.0	30.0	50.0	51.1
525 HHR	352	480	660	168 960	-	-	-	-	-	30.0	30.7
625 HHR	352	576	792	202 752	-	-	-	-	-	25.0	25.6
VGA	640	480	1 200	307 200	-	-	-	-	-	-	16.9
525 4SIF	704	480	1 320	337 920	-	-	-	-	-	-	15.3
525 SD	720	480	1 350	345 600	-	-	-	-	-	-	15.0
4CIF	704	576	1 584	405 504	-	-	-	-	-	-	12.8
625 SD	720	576	1 620	414 720	-	-	-	-	-	-	12.5
SVGA	800	600	1 900	486 400	-	-	-	-	-	-	-
XGA	1024	768	3 072	786 432	-	-	-	-	-	-	-
720p HD	1280	720	3 600	921 600	-	-	-	-	-	-	-
4VGA	1280	960	4 800	1 228 800	-	-	-	-	-	-	-
SXGA	1280	1024	5 120	1 310 720	-	-	-	-	-	-	-
525 16SIF	1408	960	5 280	1 351 680	-	-	-	-	-	-	-
16CIF	1408	1152	6 336	1 622	-	-	-	-	-	-	-

				016							
<b>4SVGA</b>	<b>1600</b>	<b>1200</b>	7 500	1 920 000	-	-	-	-	-	-	-
<b>1080 HD</b>	<b>1920</b>	<b>1088</b>	8 160	2 088 960	-	-	-	-	-	-	-
<b>2Kx1K</b>	<b>2048</b>	<b>1024</b>	8 192	2 097 152	-	-	-	-	-	-	-
<b>4XGA</b>	<b>2048</b>	<b>1536</b>	12 288	3 145 728	-	-	-	-	-	-	-
<b>16VGA</b>	<b>2560</b>	<b>1920</b>	19 200	4 915 200	-	-	-	-	-	-	-
<b>3616x1536 (2.35:1)</b>	<b>3616</b>	<b>1536</b>	21 696	5 554 176	-	-	-	-	-	-	-
<b>3672x1536 (2.39:1)</b>	<b>3680</b>	<b>1536</b>	22 080	5 652 480	-	-	-	-	-	-	-
<b>4Kx2K</b>	<b>4096</b>	<b>2048</b>	32 768	8 388 608	-	-	-	-	-	-	-
<b>4096x2304 (16:9)</b>	<b>4096</b>	<b>2304</b>	36 864	9 437 184	-	-	-	-	-	-	-

表 6 最大帧率举例（续）

级数:					3	3.1	3.2	4	4.1	4.2
最大帧尺寸(宏块):					1 620	3 600	5 120	8 192	8 192	8 192
最大宏块数/s					40 500	108 000	216 000	245 760	245 760	589 824
最大帧尺寸(样点):					414 720	921 600	1 310 720	2 097 152	2 097 152	2 097 152
最大样点数/s					10 368 000	27 648 000	55 296 000	62 914 560	62 914 560	125 829 120
格式	亮度 宽度	亮度 高度	总宏 块数	亮度样 点数						
SQCIF	128	96	48	12 288	172.0	172.0	172.0	172.0	172.0	172.0
QCIF	176	144	99	25 344	172.0	172.0	172.0	172.0	172.0	172.0
QVGA	320	240	300	76 800	135.0	172.0	172.0	172.0	172.0	172.0
525 SIF	352	240	330	84 480	122.7	172.0	172.0	172.0	172.0	172.0
CIF	352	288	396	101 376	102.3	172.0	172.0	172.0	172.0	172.0
525 HHR	352	480	660	168 960	61.4	163.6	172.0	172.0	172.0	172.0
625 HHR	352	576	792	202 752	51.1	136.4	172.0	172.0	172.0	172.0
VGA	640	480	1 200	307 200	33.8	90.0	172.0	172.0	172.0	172.0
525 4SIF	704	480	1 320	337 920	30.7	81.8	163.6	172.0	172.0	172.0
525 SD	720	480	1 350	345 600	30.0	80.0	160.0	172.0	172.0	172.0
4CIF	704	576	1 584	405 504	25.6	68.2	136.4	155.2	155.2	172.0
625 SD	720	576	1 620	414 720	25.0	66.7	133.3	151.7	151.7	172.0
SVGA	800	600	1 900	486 400	-	56.8	113.7	129.3	129.3	172.0
XGA	1024	768	3 072	786 432	-	35.2	70.3	80.0	80.0	160.0
720p HD	1280	720	3 600	921 600	-	30.0	60.0	68.3	68.3	136.5
4VGA	1280	960	4 800	1 228 800	-	-	45.0	51.2	51.2	102.4
SXGA	1280	1024	5 120	1 310 720	-	-	42.2	48.0	48.0	96.0
525 16SIF	1408	960	5 280	1 351 680	-	-	-	46.5	46.5	93.1

<b>16CIF</b>	<b>1408</b>	<b>1152</b>	6 336	1 622 016	-	-	-	38.8	38.8	77.6
<b>4SVGA</b>	<b>1600</b>	<b>1200</b>	7 500	1 920 000	-	-	-	32.8	32.8	65.5
<b>1080 HD</b>	<b>1920</b>	<b>1088</b>	8 160	2 088 960	-	-	-	30.1	30.1	60.2
<b>2Kx1K</b>	<b>2048</b>	<b>1024</b>	8 192	2 097 152	-	-	-	30.0	30.0	60.0
<b>4XGA</b>	<b>2048</b>	<b>1536</b>	12 288	3 145 728	-	-	-	-	-	-
<b>16VGA</b>	<b>2560</b>	<b>1920</b>	19 200	4 915 200	-	-	-	-	-	-
<b>3616x1536 (2.35:1)</b>	<b>3616</b>	<b>1536</b>	21 696	5 554 176	-	-	-	-	-	-
<b>3672x1536 (2.39:1)</b>	<b>3680</b>	<b>1536</b>	22 080	5 652 480	-	-	-	-	-	-
<b>4Kx2K</b>	<b>4096</b>	<b>2048</b>	32 768	8 388 608	-	-	-	-	-	-
<b>4096x2304 (16:9)</b>	<b>4096</b>	<b>2304</b>	36 864	9 437 184	-	-	-	-	-	-

表 7 最大帧率举例（续）

级数:					5	5.1
最大帧尺寸（宏块）:					21 696	36 864
最大宏块数/s					589 824	983 040
最大帧尺寸（样点）:					5 554 176	9 437 184
最大样点数/s					150 994 944	251 658 240
格式	亮度 宽度	亮度 高度	总宏 块数	亮度样 点数		
SQCIF	128	96	48	12 288	172.0	172.0
QCIF	176	144	99	25 344	172.0	172.0
QVGA	320	240	300	76 800	172.0	172.0
525 SIF	352	240	330	84 480	172.0	172.0
CIF	352	288	396	101 376	172.0	172.0
525 HHR	352	480	660	168 960	172.0	172.0
625 HHR	352	576	792	202 752	172.0	172.0
VGA	640	480	1 200	307 200	172.0	172.0
525 4SIF	704	480	1 320	337 920	172.0	172.0
525 SD	720	480	1 350	345 600	172.0	172.0
4CIF	704	576	1 584	405 504	172.0	172.0
625 SD	720	576	1 620	414 720	172.0	172.0
SVGA	800	600	1 900	486 400	172.0	172.0
XGA	1024	768	3 072	786 432	172.0	172.0
720p HD	1280	720	3 600	921 600	163.8	172.0
4VGA	1280	960	4 800	1 228 800	122.9	172.0
SXGA	1280	1024	5 120	1 310 720	115.2	172.0
525 16SIF	1408	960	5 280	1 351 680	111.7	172.0
16CIF	1408	1152	6 336	1 622	93.1	155.2

				016		
<b>4SVGA</b>	<b>1600</b>	<b>1200</b>	7 500	1 920 000	78.6	131.1
<b>1080 HD</b>	<b>1920</b>	<b>1088</b>	8 160	2 088 960	72.3	120.5
<b>2Kx1K</b>	<b>2048</b>	<b>1024</b>	8 192	2 097 152	72.0	120.0
<b>4XGA</b>	<b>2048</b>	<b>1536</b>	12 288	3 145 728	48.0	80.0
<b>16VGA</b>	<b>2560</b>	<b>1920</b>	19 200	4 915 200	30.7	51.2
<b>3616x1536 (2.35:1)</b>	<b>3616</b>	<b>1536</b>	21 696	5 554 176	27.2	45.3
<b>3672x1536 (2.39:1)</b>	<b>3680</b>	<b>1536</b>	22 080	5 652 480	26.7	44.5
<b>4Kx2K</b>	<b>4096</b>	<b>2048</b>	32 768	8 388 608	-	30.0
<b>4096x2304 (16:9)</b>	<b>4096</b>	<b>2304</b>	36 864	9 437 184	-	26.7