南京邮电学院

硕士学位论文摘要

学科　　：工科

专业　　：信号与信息处理

研究方向　：多媒体通信

作者　　：周恒 (zhou_heng@yahoo.com)

导师　　：李晓飞 副教授

题目　　：H.264 的研究与软件实现

英文题目　：The Study and Software Implementation of H.264

关键字　：H.264，　AVC，　编码器，　MPEG-4

Keywords　：H.264,　AVC,　Encoder，　MPEG-4

摘要

H.264 是由 ITU-T 的 VCEG 和 ISO 的 MPEG 联合组成的 JVT 共同开发的下一代视频编码国际标准。该标准将提供更高的编码效率和更好的图像质量。

H.264 编码效率的提高是多项技术共同作用的结果，它们包括帧内预测，整数变换，不同的预测块大小，多个参考帧，更精细的预测精度，自适应算术编码，Deblocking filter 以及基于率失真优化的模式选择算法等。

本文详细介绍了 H.264 的主要技术，并用软件实现了 H.264 编码器。本文主要分为三个部分：（一）简单介绍 H.264 的主要特点，（二）详细阐述并推导 H.264 编码器所使用的部分算法，（三）介绍软件 H.264 编码器并对编码结果做出分析。（四）介绍了一种帧内预测模式选择的快速算法

Abstract

The applications including broadcast television and home entertainment and many more were made possible by the standardization of video compression technology. Today, Joint Video Team(JVT) formed including experts from both ITU-T Video Coding Experts Group(VCEG) and ISO Motion Picture Experts Group(MPEG), are now in the final stages of developing a new standard that promises to significantly outperform any other existing ones such as MPEG-4 and H.263, providing better compression of video images together with a range of features supporting high-quality, low-bitrate streaming video. The "official" title of the new standard is Advanced Video Coding(AVC); however, with its former name being H.26L, it will be widely known as its ITU document number, H.264, like the traditional and conventional working titles of its ancestors.

This thesis introduces in detail the features of H.264, as well as its software implementation. Although H.264 follows the hybrid coding scheme in common with earlier standards, improvements are achieved through the inclusion of a number of new features that distinguish itself from all existing ones, including Intra Prediction, Integer transform, CABAC, etc., which are naturally what this article mainly focuses on.

In the first chapter, I introduce briefly the coding element of H.264 that provides the majority of the dramatic improvement in compression efficiency, in relation to prior video coding standards, along with some features of error resilience and network adaptability.

In the second chapter, a systematical analyses and detailed derivation of some important H.264 encoder design algorithms are provided, such as intra prediction, integer transform, quantization and entropy coding.

In the third chapter, the implementation of H.264 encoder is illustrated and the coding performance is compared using R-D curves. At the end of the chapter, I make some analyses of problems existing in the encoder.

In the last chapter, a fast mode decision algorithm for H.264 intra prediction is presented. Based on edge direction detection, this algorithm can significantly reduce the time in calculation.

目录

引言

随着半导体技术在处理能力及存储容量上的进步，以及视频编码算法的发展，人们希望利用这些条件开发出一种性能更好的视频编码国际标准。在此需求下，ITU-T 的视频编码专家组（VCEG）和 ISO/IEC 的活动图像专家组（MPEG）组成了一个联合视频小组（JVT），着手制订一个比现有任何标准压缩性能都要高出很多的崭新的标准[1-3]，这个标准在 ITU-T 中称为 H.264，在 ISO/IEC 中称为高级视频编码（AVC），并且编入 MPEG-4，成为其第 10 部分。由于本人偏好，在本文中将该标准称为 H.264。

为了跟上视频编码技术的发展，本文对 H.264 做了较深入的研究，并参考文献[4]，以软件形式实现了 H.264 编解码器。由于目前国内尚没有 H.264 的产品，所以作者希望本文能对国内开展 H.264 的研究与开发起到一定的促进作用。

本文的章节安排如下：

第一章 介绍 H.264 标准的特点，特别是它能够提高编码性能的一些特点。

第二章 对 H.264 的编码技术做了较深入的描述，主要集中在作者做过研究的几个方面。

第三章 介绍了 H.264 的软件实现，并对编码结果做了分析。

第四章 介绍了一种基于图像边缘方向检测的快速帧内预测模式选择算法，并通过实验与常规算法做了比较。

由于 H.264 是一个较新的标准，为了叙述准确，文中一些重要的术语采用英语表达。

第一章　H.264 介绍

本章首先简单介绍一下 H.264 的制订过程和应用场合，然后列出了 H.264 的主要特点，特别是那些能够产生显著压缩性能的特点，从这些特点中读者可以初步认识到 H.264 在性能上比目前所有的视频编码标准都要优越的原因。

1.1 H.264 的制定过程和应用场合

在制订完最初的 H.263 标准之后，ITU-T 的视频编码专家组（VCEG）将开发工作分为两部分[16]：一部分称之为“短期（short-tem）”计划，目的是给 H.263 增加一些新的特性（这一计划开发出了 H.263+和 H.263++），另一部分被称为“长期（long-term）”计划，其最初的目标就是要制定出一个比当时其他的视频编码标准效率提高一倍的新标准[5,6]。这一计划在 1997 年开始，其成果就是作为 H.264 前身的 H.26L（起初叫 H.263L[7]）。在将近 2001 年底，由于 H.26L 优越的性能，ISO/IEC 的 MPEG 专家组加入到 VCEG 中来，共同成立了联合视频小组(JVT)[9]，接管了 H.26L 的开发工作。这个组织的目标是：

“研究新的视频编码算法，其目标是在性能上要比以往制定的最好的标准提高很多。”[8]

这一标准目前还在制订过程之中，其草案不断在进行修改[1-3]，但主要内容基本上已经确定下来，预计 2003 年年底之前可以正式成为国际标准，目前最新的版本是 2003 年 3 月在泰国 Pattaya 举行的 JVT 第 7 次会议上通过的[3]。由于该标准是由两个不同的组织共同制定的，因此这一新标准有两个不同的名称：在 ITU-T 中，它的名字叫 H.264；而在 ISO/IEC 中，它被称为 MPEG-4 的第 10 部分：高级视频编码（AVC）。

H.264 的应用场合相当广泛，包括可视电话（固定或移动）、实时视频会议系统、视频监控系统以及因特网视频传输，多媒体信息存储等[3,6]。

目前在国际上，加拿大的 UB Video 公司开发出了一套基于 TMS320C64x 系列的 H.26L 实时视频通信系统[10]，它可以在 160kbps 的码率下获得与 H.263+在 320kbps 下相同的图像质量。另一家加拿大的公司 VideoLocus 通过在系统中插入

一块基于 FPGA 的硬件扩展卡，在 P4 平台上实现了 H.264 的实时编解码[11]。

1.2 H.264 的特点

H.264 在编码框架上还是沿用以往的 MC-DCT 结构，即运动补偿加变换编码的混合（hybrid）结构，因此它保留了一些先前标准的特点，如不受限制的运动矢量（unrestricted motion vectors），对运动矢量的中值预测（median prediction）等。然而，以下介绍的技术使得 H.264 比之前的视频编码标准在性能上有了很大的提高。应当说明的是，这个提高不是单靠某一项技术实现的，而是由各种不同的技术带来的小的性能改进共同产生的。

1.2.1 帧内预测

对 I 帧的编码是通过利用空间相关性而非时间相关性实现的。以前的标准只利用了一个宏块（macroblock）内部的相关性，而忽视了宏块之间的相关性，所以一般编码后的数据量较大。为了能进一步利用空间相关性，H.264 引入了帧内预测以提高压缩效率。简单地说，帧内预测编码就是用周围邻近的像素值来预测当前的像素值，然后对预测误差进行编码。这种预测是基于块的，对于亮度分量（luma），块的大小可以在 16x16 和 4x4 之间选择，16x16 块有 4 种预测模式，4x4 块有 9 种预测模式；对于色度分量（chroma），预测是对整个 8x8 块进行的，有 4 种预测模式。除了 DC 预测外，其他每种预测模式对应不同方向上的预测。

1.2.2 帧间预测

与以往的标准一样，H.264 使用运动估计和运动补偿来消除时间冗余，但是它具有以下五个不同的特点：

（一）预测时所用块的大小可变

由于基于块的运动模型假设块内的所有像素都做了相同的平移，在运动比较剧烈或者运动物体的边缘处这一假设会与实际出入较大，从而导致较大的预测误差[12]，这时减小块的大小可以使假设在小的块中依然成立。另外小的块所造成的块效应相对也小，所以一般来说小的块可以提高预测的效果。

为此，H.264 一共采用了 7 种方式对一个宏块进行分割，每种方式下块的大小和形状都不相同，这就使得编码器可以根据图像的运动情况选择最好的预测模式。

与仅使用 16x16 块进行预测相比，使用不同大小和形状的块可以使码率节省 15%以上[10]。

（二）更精细的预测精度

在 H.264 中，Luma 分量的运动矢量（MV）使用 1/4 像素精度。Chroma 分量的 MV 由 luma MV 导出，由于 chroma 分辨率是 luma 的一半（对 4:2:0），所以其 MV 精度将为 1/8，这也就是说 1 个单位的 chroma MV 所代表的位移仅为 chroma 分量取样点间距离的八分之一。

如此精细的预测精度较之整数精度可以使码率节省超过 20%[10]。

（三）多参考帧（multiple reference frames）

H.264 支持多参考帧预测，即可以有多于一个（最多 5 个）的在当前帧之前解码的帧可以作为参考帧产生对当前帧的预测（motion-compensated prediction）[13]。这适用于视频序列中含有周期性运动的情况。采用这一技术，可以改善运动估计（ME）的性能，提高 H.264 解码器的错误恢复能力，但同时也增加了缓存的容量以及编解码器的复杂性。不过，正如引言中所说，H.264 的提出是基于半导体技术的飞速发展，因此这两个负担在不久的将来会变得微不足道。

较之只使用一个参考帧，使用 5 个参考帧可以节省码率 5-10%[10]。

（四）Deblocking Filter

Deblocking Filter 可以译作抗块效应滤波器，它的作用是消除经反量化和反变换后重建图像中由于预测误差产生的块效应，即块边缘处的像素值跳变，从而一来改善图像的主观质量，二来减少预测误差。H.264 中的 Deblocking Filter 还能够根据图像内容做出判断，只对由于块效应产生的像素值跳变进行平滑，而对图像中物体边缘处的像素值不连续给予保留，以免造成边缘模糊。与以往的 Deblocking Filter 不同的是，经过滤波后的图像将根据需要放在缓存中用于帧间预测，而不是仅仅在输出重建图像时用来改善主观质量，也就是说该滤波器位于解码环中而非解码环的输出外，因而它又称做 Loop Filter。需要注意的是，对于帧内预测，使用的是未经过滤波的重建图像。

（五）SP Slice

见 1.2.5 小节。

1.2.3 整数变换

H.264 对帧内或帧间预测的残差（residual）进行 DCT 变换编码。为了克服浮点运算带来的硬件设计复杂，更重要的是舍入误差造成的编码器和解码器之间不匹配（mismatch）的问题，新标准对 DCT 的定义做了修改，使得变换仅用整数加减法和移位操作即可实现，这样在不考虑量化影响的情况下，解码端的输出可以准确地恢复编码端的输入。当然这样做的代价是压缩性能的略微下降。此外，该变换是针对 4x4 块进行的，这也有助于减少块效应。

为了进一步利用图像的空间相关性，在对 chroma 的预测残差和 16x16 帧内预测的预测残差进行上述整数 DCT 变换之后，标准还将每个 4x4 变换系数块中的 DC 系数组成 2x2 或 4x4 大小的块，进一步做哈达玛（Hadamard）变换。

1.2.4 熵编码

对于 Slice 层以上的数据，H.264 采用 Exp-Golomb 码，这是一种没有自适应能力的 VLC。而对于 Slice 层（含）以下的数据，如果是残差，H.264 有两种熵编码的方式：基于上下文的自适应变长码（Context-based Adaptive Variable Length Coding, CAVLC）和基于上下文的自适应二进制算术编码（Context-based Adaptive Binary Arithmetic Coding, CABAC）；如果不是残差，H.264 采用 Exp-Golomb 码或 CABAC 编码，视编码器的设置而定。

（一）CAVLC

VLC 的基本思想就是对出现频率大的符号使用较短的码字，而出现频率小的符号采用较长的码字。这样可以使得平均码长最小[14]。

在 CAVLC 中，H.264 采用若干 VLC 码表，不同的码表对应不同的概率模型。编码器能够根据上下文，如周围块的非零系数数或系数的绝对值大小，在这些码表中自动地选择，最大程度地与当前数据的概率模型匹配，从而实现了上下文自适应的功能。

（二）CABAC

算术编码是一种高效的熵编码方案，其每个符号所对应的码长被认为是分

数。由于对每一个符号的编码都与以前编码的结果有关，所以它考虑的是信源符号序列整体的概率特性，而不是单个符号的概率特性，因而它能够更大程度地逼近信源的极限熵H_∞，降低码率。

为了绕开算术编码中无限精度小数的表示问题以及对信源符号概率进行估计，现代的算术编码多以有限状态机的方式实现，H.264 的 CABAC 便是一个例子，其他的例子还有 JPEG2000[15]。在 CABAC 中，每编码一个二进制符号，编码器就会自动调整对信源概率模型（用一个“状态”来表示）的估计，随后的二进制符号就在这个更新了的概率模型基础上进行编码。这样的编码器不需要信源统计特性的先验知识，而是在编码过程中自适应地估计。显然，与 CAVLC 编码中预先设定好若干概率模型的方法比较起来，CABAC 有更大的灵活性，可以获得更好的编码性能——大约 10%码率的降低[10]。

以上介绍的特点都是用来提高 H.264 的编码性能的，此外 H.264 还具有很好的错误恢复能力（error resilience）和网络适应性（network adaptability），下面介绍其中的一些特点。

1.2.5 SP Slice

SP Slice 的引入最初起源于文献[17]，它主要的目的是用于不同码流的切换（switch），此外也可用于码流的随机访问、快进快退和错误恢复。这里所说的不同码流是指在不同比特率限制下对同一信源进行编码所产生的码流。设切换前传输码流中的最后一帧为 A_1，切换后的目标码流第一帧为 B_2（假设是 P 帧），由于 B_2 的参考帧不存在，所以直接切换显然会导致很大的失真，而且这种失真还会向后传递。一种简单的解决方法就是传输帧内编码的 B_2，但是一般 I 帧的数据量很大，这种方法会造成传输码率的陡然增加。根据前面的假设，由于是对同一信源进行编码，尽管比特率不同，但切换前后的两帧必然有相当大的相关性，所以编码器可以将 A_1 作为 B_2 的参考帧，对 B_2 进行帧间预测，预测误差就是 SP Slice，然后通过传递 SP Slice 完成码流的切换。与常规 P 帧不同的是，生成 SP Slice 所进行的预测是在 A_1 和 B_2 的变换域中进行的。SP Slice 要求切换后 B_2 的图像应和直接传送目标码流时一样。显然，如果切换的目标是毫不相关的另一码流，SP

Slice 就不适用了。

1.2.6 灵活的宏块排序

灵活的宏块排序（flexible macroblock ordering, FMO），是指将一幅图像中的宏块分成几个组，分别独立编码，某一个组中的宏块不一定是在常规的扫描顺序下前后连续，而可能是随机地分散在图像中的各个不同位置。这样在传输时如果发生错误，某个组中的某些宏块不能正确解码时，解码器仍然可以根据图像的空间相关性依靠其周围正确译码的像素对其进行恢复。

1.3 MPEG-4 杀手

通过上面的介绍，毋庸置疑，H.264 在压缩性能上要比其他的标准优越，甚至包括 MPEG-4[18]。众所周知，MPEG-4 最大的特点就是面向对象的编码，对象概念的提出是具有先进性的，在对象已经提取出来的条件下确实能够获得很高的压缩比，但是如何提取对象成为摆在人们面前的一大难题。我个人认为一个真正的对象提取算法应该是像人一样具有智能，能够像人一样进行思维并且是能够学习的，而目前的技术根本达不到这点，虽然有大量的文献介绍对象提取的方法，但我认为这些只是权宜之计，充其量只是往正确的方向上迈出的一小步。正因为如此，我认为 MPEG-4 面向对象编码的思想过于超前。而 ITU-T 的 VCEG 放弃了对象这一不现实的概念，与目前科学技术的发展水平相适应，提出了 H.264（H.26L）视频编码标准，我认为是难能可贵的，更重要的是它同样实现了 MPEG-4 面向对象编码的目标之一——高压缩比。目前 H.264 的主要不足就是复杂度大，但随着技术的不断进步，特别是半导体技术的发展，芯片的处理能力和存储器的容量都将会有很大的提高，所以我认为在今后 H.264 必然焕发出蓬勃的生命力，取代 MPEG-4 成为三年后市场的主角。

第二章　H.264 编码技术

本章深入讲述 H.264 的编码技术，主要集中在作者做过研究的几个方面。由于篇幅所限，一些技术上的细节请查考文献[3]。

2.1 概述

视频信号的数据量是很大的，为了达到高效的压缩，必须充分利用各种冗余，一般来讲，视频序列里的冗余包括两类[12]，一类是统计冗余，它包含[19]1）频谱冗余，指色彩分量之间的相关性；2）空间冗余；3）时间冗余，这是视频压缩区别于静止图像压缩的根本点，视频压缩主要利用时间冗余来实现大的压缩比。第二类是视觉生理冗余，这是由于人类的视觉系统（HVS）特性造成的，比如人眼对色彩分量没有对亮度分量敏感，对图像高频（即细节）处的噪声不敏感等。

针对这些冗余，视频压缩算法采用了不同的方法加以利用，但主要的考虑集中在空间冗余和时间冗余上。与以前的标准类似，H.264 也采用了所谓的混合（hybrid）结构，即对空间冗余和时间冗余分别进行处理[36]。对空间冗余，标准通过变换及量化达到消除的目的，这样编码的帧叫做 I 帧；而时间冗余则是通过帧间预测，即运动估计和补偿来去除的，这样编码的帧叫做 P 帧或 B 帧。

与以前的标准不同的是，H.264 在编码 I 帧时，采用了帧内预测，然后对预测误差进行编码。这样就充分利用了空间相关性，提高了编码效率。H.264 的帧内编码框图如图 2-1 所示。注意，该框图反映的仅仅是帧内编码一个大概的轮廓与流程，具体内容将在随后内容中详细讲述。

H.264 帧内预测以 16x16 的宏块为基本单位。首先，编码器将与当前宏块同一帧的邻近像素作为参考，产生对当前宏块的预测值（见 2.2 节），然后对预测残差进行变换与量化（见 2.3 节），再对变换与量化后的结果做熵编码（见 2.4 节）。熵编码的结果就可以形成码流了。由于在解码器端能够得到的参考数据都是经过反变换与反量化后的重建图像，所以为了使编解码一致，编码器端用于预测的参考数据就和解码器端一样，也是经过反变换与反量化后的重建图像。需要注意的一点是，用于帧内预测的这些参考数据不需要经过 Deblocking Filter 滤波，

这与帧间编码的参考图像是不同的。

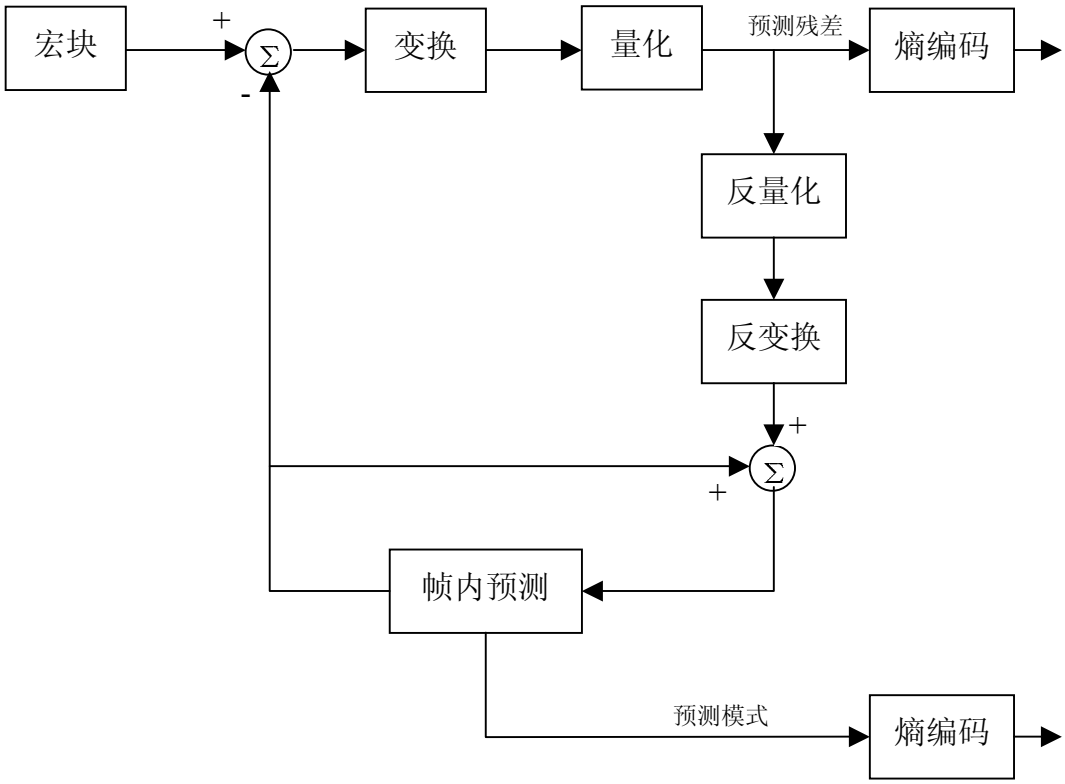


图 2-1 帧内编码框图

2.2 帧内预测

帧内预测的目的是生成对当前宏块的预测值。一个宏块由一个 16x16 的 luma 分量和两个 8x8 的 chroma 分量构成，luma 块有两类帧内预测方式，按标准中的记号表示为：Intra_16x16 和 Intra_4x4；而两个 chroma 分量则采用相同的预测方式。Intra_16x16 是对整个 16x16 大小的 luma 进行预测，一般用于图像比较平坦的区域，共有 4 种预测方式。而 Intra_4x4 方式是将 16x16 大小的 luma 划分为 16 个 4x4 大小的亮度块，然后对每个 4x4 大小的块进行预测，共有 9 种预测方式。对于 chroma 分量（Cr 和 Cb），预测是对整个 8x8 块进行的，共 4 种预测方式。几种帧内预测的模式见图 2-2。

帧内预测 { luma { Intra_16x16 ：4种
Intra_4x4 ：9种
chroma(Cr,Cb) ：4种

图 2-2 帧内预测的模式

2.2.1 Intra_16x16 帧内预测模式

Intra_16x16 帧内预测模式根据与当前宏块邻近的 33 个像素来生成 luma 分量的预测数据，共有 4 种预测方式：垂直（vertical）、水平（horizontal）、DC 和平面（plane），如图 2-3 所示。图中灰色部分代表当前宏块，绿色部分代表作为参考的邻近像素，它们分别来自当前宏块的上方（以 H 表示）、左方（以 V 表示）和左上方，因为这些宏块在编码顺序上位于当前宏块之前，所以用它们来预测是合理的。

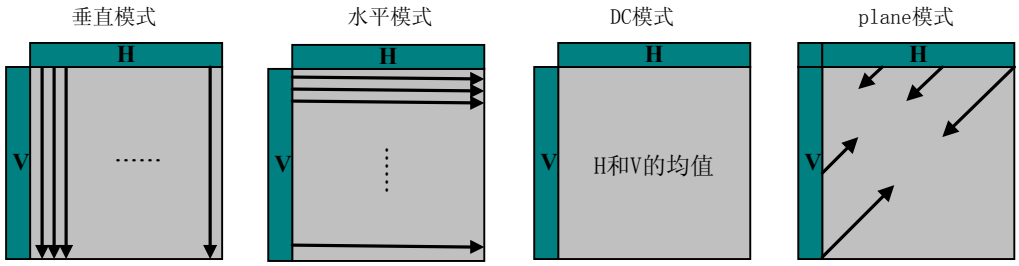


图 2-3 Intra_16x16 帧内预测模式

在进行预测之前，首先要判断这些邻近像素是否可用（available），如果这些像素不可用，例如邻近像素所在的宏块位于其它 Slice 之中或当前宏块位于图像边缘时，某些预测模式就用不起来。具体判断的细节请参考标准。

对于垂直模式，如果 H 可用的话，预测值即为 H，否则不能使用此模式。

对于水平模式，如果 V 可用的话，预测值即为 V，否则不能使用此模式。

对于 DC 模式，如果 H 和 V 都可用，就用这 32 个像素的均值作为预测值，如果只有 H 或 V 可用，就用这 16 个像素的均值作为预测值，如果 H 和 V 都不可用，例如当前宏块位于一个 Slice 的开头，则预测值为 128。

对于 plane 模式，要求必须所有的 33 个邻近像素都可用。这种方式实质上就是利用 H 和 V，顺着图 2-3 箭头所指的方向做外插（extrapolation）。为便于叙述，引入一个坐标系，其中横向为 x 轴，纵向为 y 轴，定义当前宏块左上角像素

的坐标为(0,0)。用 $p(x,y)$表示位于坐标(x,y)处的 33 个邻近像素值，其中 H 对应 $p(x,-1), x=0..15$，V 对应 $p(-1,y), y=0..15$，而左上角处的邻近像素值为 $p(-1,-1)$。预测值用 $\text{pred}_L(x,y), x,y=0..15$ 表示。则 plane 预测模式的步骤及公式如下：

1）计算中间变量 H 和 V：

$$H = \sum_{x=0}^{7}(x+1)*(p(8+x,-1)-p(6-x,-1))$$

$$V = \sum_{y=0}^{7}(y+1)*(p(-1,8+y)-p(-1,6-y))$$

2）计算中间变量 a,b,c：

$$a = 16*(p(-1,15)+p(15,-1))$$

$$b = (5*H+32)>>6$$

$$c = (5*V+32)>>6$$

3）计算预测值 $\text{pred}_L(x,y)$：

$$\text{pred}_L(x,y) = \text{Clip1}((a+b*(x-7)+c*(y-7)+16)>>5) \qquad x,y=0..15$$

其中 Clip1(x)代表将 x 箝位于 0 到 255 之间（含），即

$$\text{Clip1}(x) = \begin{cases} 255 & x > 255 \\ 0 & x < 0 \\ x & 0 \le x \le 255 \end{cases}$$

2.2.2 Intra_4x4 帧内预测模式

在此模式下，编码器将当前宏块 16x16 的 luma 分量划分为 16 个 4x4 的块，然后根据每个 4x4 块周围的邻近像素对该块做预测。按理说，对一块像素做预测，其上下左右的像素都应当作为参考，但由于编码顺序的原因，H.264 只选择了 13 个像素作为参考，这 13 个邻近像素与当前块的位置关系如图 2-4 中 A-M 所示。

M	A	B	C	D	E	F	G	H
I								
J								
K								
L								

图 2-4 Intra_4x4 预测的邻近像素

同 Intra_16x16 模式一样，在开始预测之前，首先需判断 A-M 这些参考像素是否可用，如果有些参考像素不可用，那么有些预测模式也就不能用了。前已提及，编码器端用于预测的参考数据是经过反变换与反量化后的重建图像，所以判断的主要依据就是看这些像素是否在当前的 4x4 块之前已经完成编码，即已经是经过重建了的图像。例如在图 2-9 中，对于宏块内序号为 3 和 11 的 4x4 块，由于它们的 E-H 参考像素所在的 4x4 块（序号分别为 4 和 12）尚未编码，所以它们的 E-H 参考像素是不可用的。另外，如果 E-H 不可用而 D 是可用的，则用 D 来替代 E-H 并将 E-H 标记为可用。

在决定了哪些参考像素可用后，就可以通过它们产生预测值了。H.264 一共定义了 9 种 Intra_4x4 预测方式，除了 DC 方式（模式 2）之外，其它 8 种都是向某一个方向上进行预测，也就是做外插。这 8 个预测方向和预测的方式如图 2-5 和图 2-6[16]所示。具体预测值如何生成请参见标准。

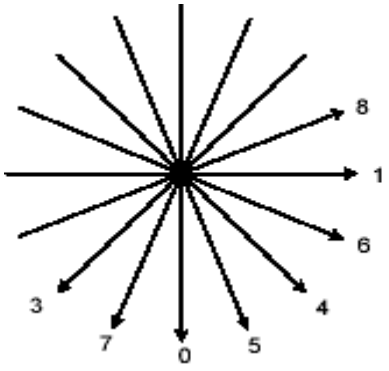


图 2-5 Intra_4x4 的 8 种预测方向

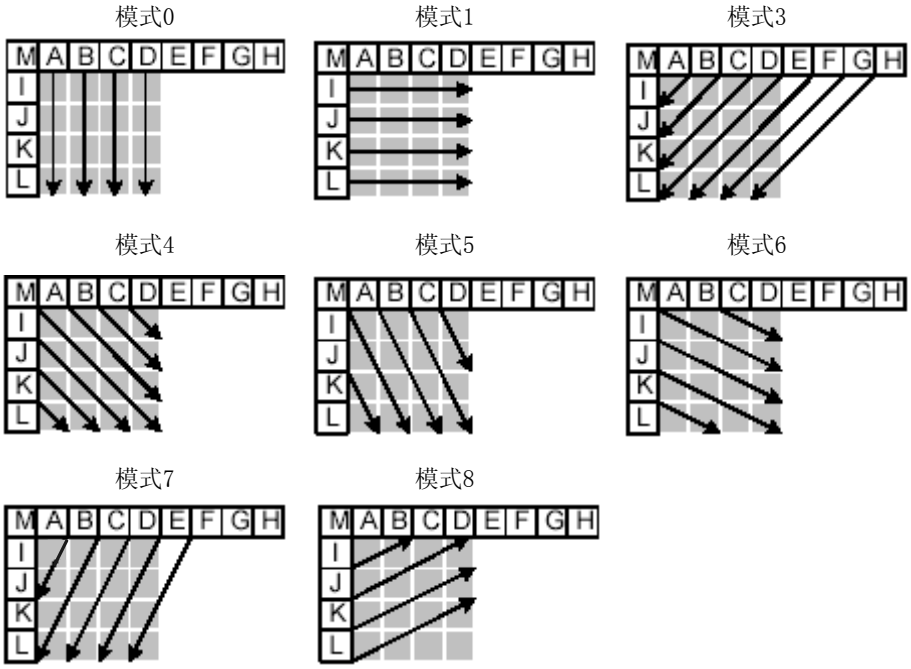


图 2-6 Intra_4x4 的 8 种方向性预测

由图中可以看出，如果某一块区域内的像素值呈现出一种方向性，那么选用与这个方向相近的某个预测方式，将会达到较好的预测效果，本文后面将对这一问题做进一步研究。

作为空间相关性的一种表现，在空间上相临的块其预测模式也是相近的，所以对于 Intra_4x4 预测，H.264 并不直接编码各个块的预测模式，而是根据当前块左边和上边块的预测模式，对当前块的预测模式进行估计，只有当前块的预测模式和这个估计出来的预测模式不相同时才额外传送当前块的预测模式。

2.2.3 色差分量的帧内预测

该预测针对的是当前宏块的两个 8x8 的色差分量 Cr 和 Cb，共有 4 种模式，两个 chroma 分量采用相同的预测模式，预测对两个分量分别进行，预测的范围是整个 8x8 的色差分量。预测的参考像素是同一个 chroma 分量的周围 17 个像素，如图 2-7 所示，图中灰色部分代表当前 chroma 分量，左边、上边和左上方白色的方框代表参考像素。

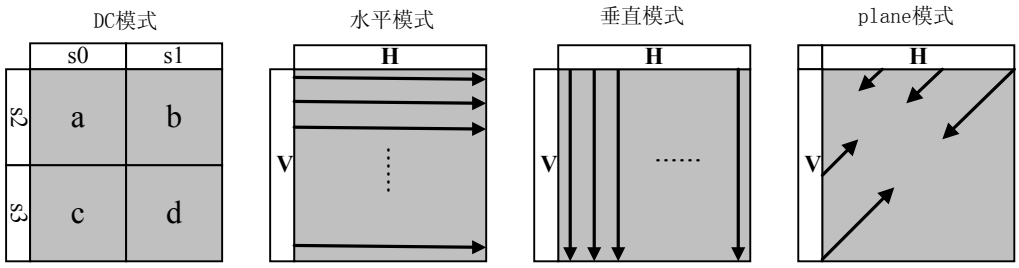


图 2-7 色差分量的 4 种帧内预测模式

预测的过程同 Intra_16x16 基本相似：首先判断这 17 个参考像素的可用性（availability），然后进行预测。这里仅说明一下 DC 模式的预测过程。

图 2-7 最左边是 DC 模式的示意图，图中 s0-s3 代表所处 4 个像素的平均值，a-b 是 4 个 4x4 的块，它们的预测值要由 s0-s3 的可用性来决定。表 2-1 显示了该模式的预测过程。在表 2-1 中，每个块的预测都从最左端的一栏开始，如果预测公式所涉及到的 s 不可用，就转至右边的一栏。例如，如果 s0 和 s3 可用而 s1 和 s2 不可用，则 a-b 的预测分别采用 Pred1, Pred3, Pred0 和 Pred2 中的公式。

表 2-1 色差分量的 DC 帧内预测模式过程

块	Pred0	Pred1	Pred2	Pred3
a	(s0+s2+4)>>3	(s0+2)>>2	(s2+2)>>2	128
b	(s1+2)>>2	(s1+2)>>2	(s2+2)>>2	128
c	(s3+2)>>2	(s0+2)>>2	(s3+2)>>2	128
d	(s1+s3+4)>>3	(s1+2)>>2	(s3+2)>>2	128

2.3 变换与量化

将图像的当前像素值与预测值相减，就形成了预测残差。残差内仍然含有空间冗余，为了消除这种冗余，通常采用变换编码[20]，即变换-量化-熵编码三步。变换并不压缩数据，它只是消除数据中的相关性，或者说将数据中的冗余（或相关性）以一种便于随后进行熵编码的方式表现出来。压缩是在熵编码步骤中完成的。此外为了进一步减少数据量，编码器还对变换后的系数进行量化，它的实质是减少数据的取值范围以减少每一个符号的熵。它会造成信息的损失，是有损编码的一个重要步骤，它也是控制图像率失真（R-D）特性的一个主要手段。在 H.264 中，变换与量化两个步骤紧密相连。

图像编码中常用的变换是 DCT，因为它在某种条件下近似于理论上最优的 K-L 变换[21]。但是如果直接采用 DCT 的定义进行变换，会带来两个问题：一个是需要进行浮点数操作，从而造成系统设计上的复杂性；第二，由于变换核都是无理数，而有限精度的浮点数不可能精确地表示无理数，再加上浮点数的运算可能会引入舍入误差，这就使得在具体实现时会导致编解码的不一致（mismatch），即反变换的输出结果和正变换的输入不一样。为了克服这些问题，H.264 采用整数 DCT 变换，使得变换操作仅用整数加减和移位操作就可以完成，这样既降低了设计复杂度，又避免了编解码 mismatch，而由此带来的编码性能的减少微乎其微。需要注意的是，此时的变换已经不是真正的 DCT，仍然称其为 DCT 变换只是为了说它是由 DCT 推导而来，且为了和另一个变换（Hadamard 变换）相区别罢了。

H.264 编码器的变换与量化过程如图 2-8 所示：

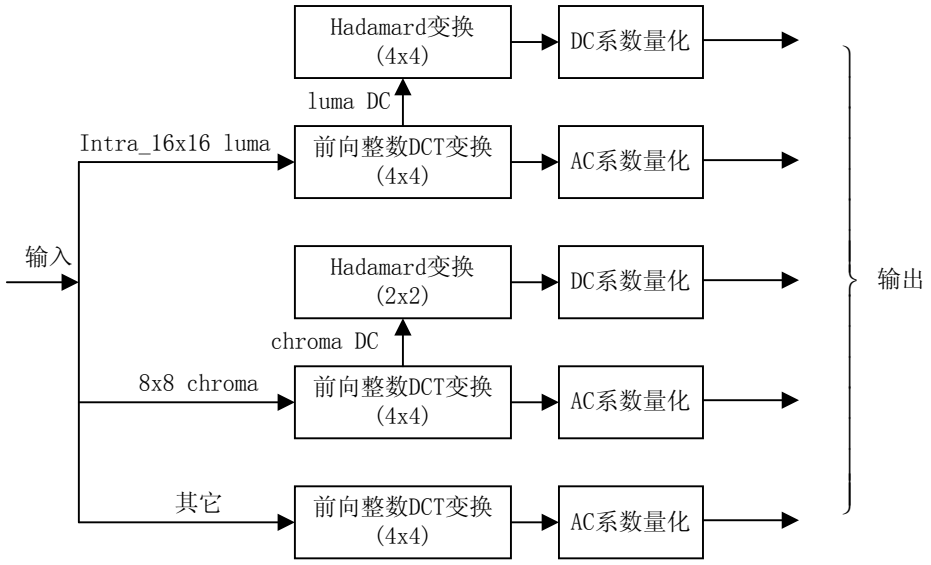


图 2-8 H.264 编码器变换与量化过程

图中输入为预测残差，输出为准备进行熵编码的数据，一共有五类。为了更大程度地利用空间冗余，对于 Intra_16x16 帧内预测模式，H.264 在对 16x16 的 luma 分量的 16 个 4x4 块进行 DCT 变换后，将每个 4x4 块的 DC 系数（还没有经过量化）提取出来，组成一个 4x4 的 luma DC 块，对其再进行 4x4 的哈达玛（Hadamard）变换。同样，对 8x8 chroma 分量的 4 个 4x4 块进行 DCT 变换后，也将每个 4x4 块的 DC 系数提取出来，组成一个 2x2 的 chroma DC 块，对其进行 2x2 的 Hadamard 变换，如图 2-9 所示。图中的数字显示的是所代表的块在码流

中的顺序。

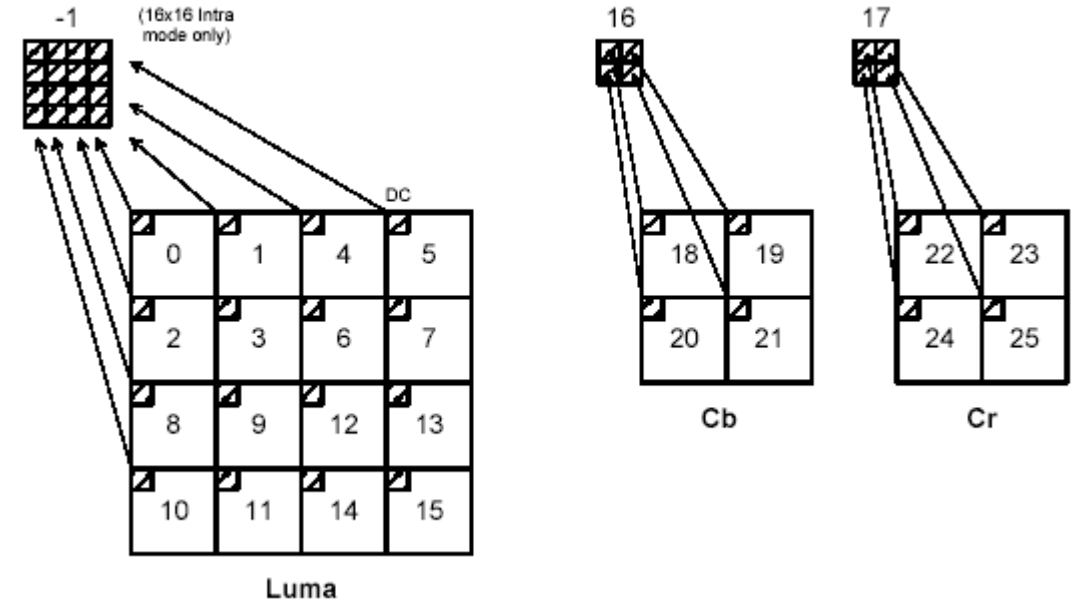


图 2-9 对 DC 系数的处理

H.264 解码器的反变换与反量化过程如图 2-10 所示：

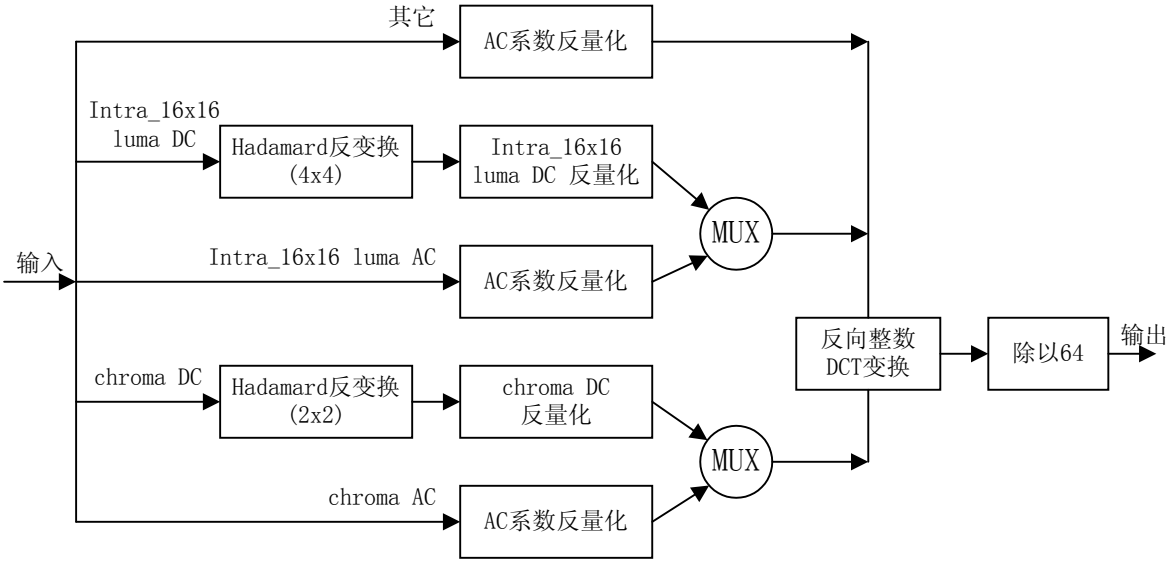


图 2-10 H.264 解码器反变换与反量化过程

图中的输入是经过解码（CAVLC 或 CABAC）后的结果，输出的数据加上预测值后成为重建图像，重建图像用于帧内预测，或经过 deblocking filter 后显示并根据需要存放于缓存中，用于帧间预测。这里有一个地方需要注意，对于 DC 系数（无论是 Intra_16x16 luma DC 还是 chroma DC），解码器是先反变换再反量化，这样做的原因在后面的内容中将做解释。MUX 是指将 DC 系数按图 2-9 装配到 AC 系数中，形成完整的 4x4 块，用于后续的反 DCT 变换。

2.3.1 变换

本节推导 H.264 中的整数 DCT 变换公式[22]。

首先介绍一个关于矩阵运算的结论：一个 $N\times N$ 的矩阵可以分解为一个对角阵和另一个矩阵相乘，进一步又可以等于两个矩阵点乘，如公式 2.1 所示。这里“点乘”是借用 Matlab 里的概念，含义为两个矩阵对应元素相乘，用符号⊗表示，显然，点乘满足交换率。

$$R=\begin{pmatrix} r_{11}s_1 & r_{12}s_1 & \cdots & r_{1N}s_1 \\ r_{21}s_2 & r_{22}s_2 & \cdots & r_{2N}s_2 \\ \vdots & \vdots & \vdots & \vdots \\ r_{N1}s_N & r_{N2}s_N & \cdots & r_{NN}s_N \end{pmatrix}=\begin{pmatrix} s_1 & & & \\ & s_2 & & \\ & & \ddots & \\ & & & s_N \end{pmatrix}\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1N} \\ r_{21} & r_{22} & \cdots & r_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ r_{N1} & r_{N2} & \cdots & r_{NN} \end{pmatrix}$$
$$=\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1N} \\ r_{21} & r_{22} & \cdots & r_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ r_{N1} & r_{N2} & \cdots & r_{NN} \end{pmatrix}\otimes\begin{pmatrix} s_1 & s_1 & \cdots & s_1 \\ s_2 & s_2 & \cdots & s_2 \\ \vdots & \vdots & \vdots & \vdots \\ s_N & s_N & \cdots & s_N \end{pmatrix} \tag{2.1a}$$

$$R=\begin{pmatrix} r_{11}s_1 & r_{12}s_2 & \cdots & r_{1N}s_N \\ r_{21}s_1 & r_{22}s_2 & \cdots & r_{2N}s_N \\ \vdots & \vdots & \vdots & \vdots \\ r_{N1}s_1 & r_{N2}s_2 & \cdots & r_{NN}s_N \end{pmatrix}=\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1N} \\ r_{21} & r_{22} & \cdots & r_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ r_{N1} & r_{N2} & \cdots & r_{NN} \end{pmatrix}\begin{pmatrix} s_1 & & & \\ & s_2 & & \\ & & \ddots & \\ & & & s_N \end{pmatrix}$$
$$=\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1N} \\ r_{21} & r_{22} & \cdots & r_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ r_{N1} & r_{N2} & \cdots & r_{NN} \end{pmatrix}\otimes\begin{pmatrix} s_1 & s_2 & \cdots & s_N \\ s_1 & s_2 & \cdots & s_N \\ \vdots & \vdots & \vdots & \vdots \\ s_1 & s_2 & \cdots & s_N \end{pmatrix} \tag{2.1b}$$

矩阵形式的 2-D DCT 定义如下：

$$Y=AXA^T=\begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & c \end{pmatrix}X\begin{pmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & c \end{pmatrix} \tag{2.2}$$

其中 $a=\frac{1}{2}$，$b=\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right)$，$c=\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right)$。

将矩阵 A 及其转置分别应用公式(2.1a)和(2.1b)，得

$$
\begin{aligned}
Y &= BCXC^T B^T \\
&= \begin{pmatrix} a & & & \\ & b & & \\ & & a & \\ & & & b \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{pmatrix} X \begin{pmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{pmatrix}\begin{pmatrix} a & & & \\ & b & & \\ & & a & \\ & & & b \end{pmatrix} \\
&= \left[\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{pmatrix} X \begin{pmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{pmatrix}\right] \otimes \begin{pmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{pmatrix}
\end{aligned}
\tag{2.3}
$$

其中

$$
d = \frac{c}{b} \tag{2.4}
$$

经计算可得，d=0.4142…，为达到整数变换的目的，令 $d = 1/2$。为使变换是正交的，即 $AA^T = I$，需要对 b 进行调整。由线性代数知识，一个矩阵为正交，必须行向量为单位向量，即各个元素的平方和为 1，由公式(2.2)中 A 的第 2 行，并将 $c = bd$ 代入，有：

$$
b^2 + c^2 + (-c)^2 + (-b)^2 = 2b^2(1 + d^2) = 1
$$

解得 $b = \sqrt{\frac{2}{5}}$。

这样我们可以得到整数 DCT 正变换的公式：

$$
Y = \left[\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{pmatrix} X \begin{pmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{pmatrix}\right] \otimes \begin{pmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{pmatrix}
\tag{2.5}
$$

式(2.5)中，虽然乘以 1/2 的操作可以用右移来实现，但这样会产生截断误差，因此，我们再一次利用公式(2.1)，将 1/2 提到矩阵外面，并与右边的点乘合并，得：

$$
\begin{aligned}
Y &= (C_f X C_f^T) \otimes E_f \\
&= \left[\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} X \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix} \right] \otimes \begin{pmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{pmatrix}
\end{aligned}
\tag{2.6}
$$

这就是 H.264 中所用到的整数变换公式，其变换核 $C_f X C_f^T$ 仅用加减法（和左移）即可以实现。而后面的点乘操作可以合并到随后的量化过程中去。

下面推导反变换公式。

由公式(2.3)的第一个等号，我们有：

$$
X = C^T (B^T Y B) C \tag{2.7}
$$

由于 B 是对角阵，由公式(2.1)，将其转化为点乘，得

$$
\begin{aligned}
X &= C_i^T (Y \otimes E_i) C_i \\
&= \begin{pmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{pmatrix} \left[Y \otimes \begin{pmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{pmatrix} \right] \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{pmatrix}
\end{aligned}
\tag{2.8}
$$

这就是 H.264 的反 DCT 变换公式，与 Y 点乘的操作与反量化合并，乘以系数 1/2 的操作由右移来实现，由于反量化后的结果足够地大，所以这里不会出现截断误差的问题，这一点下面将会讨论。

上面说过，对 Intra_16x16 luma DC 和 chroma DC 要做 Hadamard 变换，应用于 Intra_16x16 luma DC 的 4 阶 Hadamard 变换的定义如下[23]：

$$
Y_{D4} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} X_{D4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \tag{2.9}
$$

应用于 chroma DC 的 2 阶 Hadamard 变换的定义如下：

$$
Y_{D2} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} X_{D2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.10}
$$

Hadamard 变换的正变换和反变换形式一样。

由于矩阵乘法满足结合律，故公式(2.6)、(2.8-10)所表示的变换是可分离的，这样 2-D 变换就可以用 1-D 变换来实现。所谓可分离的变换，是指变换可以分做两步完成：先对需要做变换的矩阵的每一列做 1-D 变换，再对其结果的每一行做 1-D 变换，这个次序也可以反过来，先行后列。在具体实现的时候为了减少运算量，每一步可以采用蝶型算法，以公式(2.6)的第一步对 X 的第一列进行 1-D 变换为例，其运算过程如下式所示：

$$\begin{aligned} x_0^{'} &= x_0 + x_1 + x_2 + x_3 \\ x_1^{'} &= 2x_0 + x_1 - x_2 - 2x_3 \\ x_2^{'} &= x_0 - x_1 - x_2 + x_3 \\ x_3^{'} &= x_0 - 2x_1 + 2x_2 - x_3 \end{aligned} \tag{2.11}$$

其中 $x_n, n = 0..3$ 为 X 第一列的元素，$x_n^{'}, n = 0..3$ 为滤波结果。观察(2.11)，可以发现计算有很多重复，如 $x_0 + x_3$ 就同时被计算 $x_0^{'}$ 和 $x_2^{'}$ 的公式所使用，所以可以将其暂时保存起来以避免重复计算，对应的蝶型算法如下图所示。

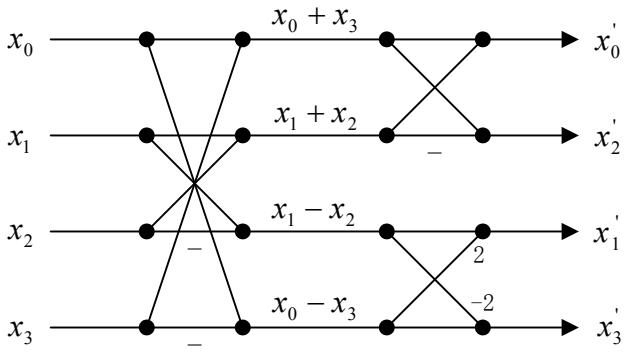


图 2-11 蝶型算法

公式(2.11)中，为了计算 $x_n^{'}, n = 0..3$，需要进行 12 次加法，4 次乘法，而图 2-11 中的蝶型算法仅需 8 次加法，2 次乘法，它利用了运算中的冗余，降低了运算量，其代价是需要额外的存储空间存放中间变量。

众所周知，利用数据中的冗余，可以压缩数据，降低码率，如游程编码。而对于一些运算，如公式(2.11)，我们也可以利用其中的冗余减低运算量。但信息论告诉我们，数据不可能无限制地无损压缩，它的码率只能逼近而不能超过信源的平均符号熵。由此我就联想到，是否一些运算，或公式，其内部也有类似于平

均符号熵这样的量，它决定了简化运算、降低运算量的下限，超过这个下限的简化运算是不可能的，或会对结果产生误差。例如上面的公式(2.11)，运用蝶型算法可以使加法和乘法次数降到 8 和 2，这是不是理论上能够达到的最小次数？如果不是，可以继续研究更好的算法，但如果已经是理论最小值，那么研究如何再进一步降低运算次数就变得毫无意义。我认为这个问题的解决将比信息论更具有理论和实际意义，因为它可以避免无谓的劳动。但是我不知道关于这一问题是否已有成果，希望对此感兴趣的读者能与作者交流。

2.3.2 量化

H.264 采用标量量化，其基本公式如下：

$$z = round\left(\frac{y}{\mathrm{QStep}}\right) \tag{2.12}$$

其中 y 为输入值，QStep 为量化步长，z 为量化后的值，*round* 代表四舍五入。

H.264 标准在常规量化的基础上又考虑了两个问题：如何将变换过程中的点乘步骤一并考虑进来；如何避免除法和浮点运算。在公式(2.6)中，变换 $C_f XC_f^T$ 的结果还需与 E_f 点乘，这一步可以与量化放在一起考虑。设 w_{ij} 为 $C_f XC_f^T$ 结果中位于第 i 行第 j 列的一个元素，则由(2.6)与(2.12)，它的量化值应为：

$$z_{ij} = round\left(w_{ij}\frac{\mathrm{PF}_f}{\mathrm{QStep}}\right) \tag{2.13}$$

其中，PF_f 为 E_f 位于 (i,j) 处的元素值（见表 2-2），QStep 为量化步长，z_{ij} 为量化后的值。

H.264 用量化参数（Quantization Parameter, QP）来索引每一个 QStep，它们之间的关系如表 2-3 所示[16]。H.264 一共有 52 个不同的量化步长，这有助于对图像质量进行更细致更灵活的控制。H.264 采用非均匀量化，其量化步长有这样一个规律：QP 每增加 6，对应的 QStep 就增加一倍。这样就相当于每个 QStep 比前一个值增加约 12.25%（$\sqrt[6]{2}-1$）。这在表 2-3 中也有所反映。

表 2-2 PF_f 的值

(i,j)	PF_f
(0,0), (2,0), (0,2), (2,2)	$a^2 = 0.25$
(1,1), (1,3), (3,1), (3,3)	$b^2/4 = 0.1$
其它	$ab/2 \approx 0.158113883$

表 2-3 量化步长

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

由于 QStep 是小数，为了避免浮点数相除，H.264 采用公式(2.14)做了近似处理：

$$\frac{\mathrm{PF}_f}{\mathrm{QStep}} = \frac{\mathrm{MF}}{2^{\mathrm{qbits}}} \tag{2.14}$$

式中 MF 为整数，而 qbits 应足够地大以使 MF 既为整数又能很好地近似等式左端，故令 qbits=15。由于舍入函数 *round* 可以用公式(2.15)的移位操作实现：

$$round(x/2^b) = (x + 2^{b-1}) >> b \tag{2.15}$$

这样(2.13)的浮点数相除就可以用公式(2.16)所示的整数乘法和移位来实现了。

$$z_{ij} = (w_{ij} \cdot \mathrm{MF} + f) >> \mathrm{qbits} \tag{2.16}$$

这里 H.264 并不像我想象的那样按式(2.15)取 $f = 2^{\mathrm{qbits}}/2$，而是对帧内编码取 $2^{\mathrm{qbits}}/3$，对帧间编码取 $2^{\mathrm{qbits}}/6$，我还没搞懂这是什么原因。

MF 可以通过公式(2.14)推导出来。考虑到 QP 每隔 6 个 QStep 就增加一倍，而 QStep 增加一倍后为了使式(2.14)仍然成立 qbits 应加 1，所以可得式(2.16)中

qbits 的计算式如下：

$$\text{qbits} = 15 + floor(\text{QP}/6) \tag{2.17}$$

其中 *floor* 函数代表向下取整。由于这个“周期性”，MF 只需计算对应 QP 从 0 到 5 的 6 个值，对于 QP>5 的情况，MF 的值和对应 QP%6 的 MF 值相同，如表 2-4 所示。对于后两栏，我算出的结果和标准中的有一些出入，我不知道是否标准对这两栏还做了什么特殊的处理。

表 2-4 MF 的值

QP	(0,0), (2,0), (0,2), (2,2)	(1,1), (1,3), (3,1), (3,3)		其它	
		我算的 MF	标准中的 MF	我算的 MF	标准中的 MF
0	13107	5243	5243	8290	8066
1	11916	4766	4660	7536	7490
2	10082	4033	4194	6377	6554
3	9362	3745	3647	5921	5825
4	8192	3277	3355	5181	5243
5	7282	2913	2893	4605	4559

下面推导反量化所需的公式。

与公式(2.12)相对应，标量反量化的基本公式如下：

$$y = z \cdot \text{QStep} \tag{2.18}$$

其中 z 为量化了的数据，y 为反量化的结果，QStep 为量化步长。

在反 DCT 变化的公式(2.8)中，Y 代表已经反量化的数据，为了进行反变换，它还需与 E_i 点乘。为简化计算，H.264 解码器将反量化与点乘的操作合在一起。假设反量化的输入端为 z_{ij}，它是刚经过 CAVLC 或 CABAC 解码的数据，由公式(2.18)与(2.8)，可得

$$w_{ij} = z_{ij} \cdot \text{QStep} \cdot \text{PF}_i \cdot 64 \tag{2.19}$$

其中 PF_i 为 E_i 中位于 (i, j) 处的元素值，由 w_{ij} 形成的矩阵 W 随后按公式(2.8)做反变换（即 $X = C_i^T W C_i$）。注意 W 不是式(2.8)中的 $Y \otimes E_i$（事实上 $Y \otimes E_i = z_{ij} \cdot \text{QStep} \cdot \text{PF}_i$），它被乘了一个因子 64。这是因为，在式(2.8)中，矩阵 C_i

含有 1/2，如果用浮点运算来实现除以 2 的操作，将不会造成任何误差，但是在实际中往往用右移来实现，对于奇数，这将会产生截断误差，为了避免截断误差，我们就用 64 乘以 $Y \otimes E_i$，以保证所有与 1/2 相乘的数都是偶数，等反变换完成之后再除以 64（结果四舍五入，采用公式(2.15)所示的移位实现），由于常数在几个矩阵相乘的过程中位于什么位置都不影响结果（即 $A(B \cdot k)C = ABC \cdot k$），所以这样做就可以避免与 1/2 相乘而产生截断误差。注意，这并不意味着最后除以 64 也不会产生误差，事实上，由于经过了量化与反量化过程，式(2.6)与式(2.8)中的两个 Y 已经不能保证完全相等，所以除以 64 四舍五入后往往会有舍入误差。

在标准中，(2.19)式中的

$$\mathrm{V} = \mathrm{QStep} \cdot \mathrm{PF}_i \cdot 64 \tag{2.20}$$

已被预先算出，同样由于 QStep 的“周期性”，标准只计算了对应 QP 从 0 到 5 的 6 个值，对于 QP>5 的情况，V 的值和对应 QP%6 的 V 值相同，这样(2.19)式最终变为

$$w_{ij} = z_{ij} \cdot \mathrm{V} \cdot 2^{floor(\mathrm{QP}/6)} \tag{2.21}$$

其中 2 的幂是为了补偿 QStep 的周期性。

式(2.16)和式(2.21)的量化与反量化，只是针对 AC 系数的，对于 DC 系数，H.264 标准又做了不同处理，现将各种量化与反量化的公式汇总于表 2-5 中，其中 $\mathrm{V}_{(0,0)}$ 表示在公式(2.20)中根据位于(0,0)处的 PF_i 计算出来的 V。公式(2.22)中，输入 w_{ij} 是按公式(2.9)做正变换的结果 Y_{D4} 中的元素；由于 H.264 对于解码后的 DC 系数是先反变换再反量化，所以公式(2.23)(2.24)中的输入 z_{ij} 是按公式(2.9)做反变换的结果 Y_{D4} 中的元素，(2.25)(2.26)的输入是按公式(2.10)做反变换的结果 Y_{D2} 中的元素。它们的输出 w_{ij} 将与(2.21)的输出一起按公式(2.8)做反变换。

从公式(2.23-26)中可以看出，由于在一个宏块中 QP 的值是固定的，所以这些反量化因子都是常数，由于常数的位置不影响矩阵相乘的结果（这在上面说明如何避免截断误差的时候也提到过），所以对于 DC 系数，先反变换再反量化与先反量化再反变换的结果是相同的。而之所以采用前者是出于所谓动态范围的考

虑，这里动态范围指的是变量的取值范围。可以看出，反量化将使变量的动态范围增大，如果先反量化再反变换，矩阵相乘的过程中可能会有某个中间变量的取值溢出，而先对动态范围较小的数据做反变换，溢出的可能性将变小。

表 2-5 H.264 量化与反量化公式

	量化	反量化
AC 系数	$z_{ij} = (w_{ij} \cdot \mathrm{MF} + f) >> \mathrm{qbits}$ (2.16)	$w_{ij} = z_{ij} \cdot \mathrm{V} \cdot 2^{floor(\mathrm{QP}/6)}$ (2.21)
Intra_16x16 luma DC	$z_{ij} = (w_{ij} \cdot \mathrm{MF} + 2f) >> (\mathrm{qbits} + 1)$ (2.22)	若 $\mathrm{QP} \geq 12$： $w_{ij} = z_{ij} \cdot \mathrm{V}_{(0,0)} \cdot 2^{floor(\mathrm{QP}/6)-2}$ (2.23) 若 QP<12： $w_{ij} = \left(z_{ij} \cdot \mathrm{V}_{(0,0)} + 2^{1-floor(\mathrm{QP}/6)}\right) >> (2 - floor(\mathrm{QP}/6))$ (2.24)
chroma DC		若 $\mathrm{QP} \geq 6$： $w_{ij} = z_{ij} \cdot \mathrm{V}_{(0,0)} \cdot 2^{floor(\mathrm{QP}/6)-1}$ (2.25) 若 QP<6 $w_{ij} = \left(z_{ij} \cdot \mathrm{V}_{(0,0)}\right) >> 1$ (2.26)

2.4 熵编码

在完成变化与量化之后，这些数据就送往熵编码器，完成整个变换编码的最后一步。H.264 一共有 3 种熵编码：(1)Exp-Golomb 码（Exponential Golomb codes）；(2)CAVLC （Context-based Adaptive Variable Length Coding）；(3)CABAC（Context-based Adaptive Binary Arithmetic Coding）。H.264 对这三种码的使用范围做了规定：(1)不出现在残差数据中；(2)仅出现在残差数据中；(3)仅出现在 Slice

层以下（从 Slice_data 开始）的数据中。(1)和(2)都是采用查表方式，但是(1)的表是固定的，而(2)在编码过程中会根据周围宏块以及在之前编码的数据信息，选择不同的表，从而具有上下文自适应功能。(3)属于自适应算术编码，能够获得比(2)更好的压缩性能和自适应能力。

2.4.1 Exp-Golomb 码

该编码最早在文献[24]中提出，后由于其易于用 DSPs 实现及更好的错误恢复能力被 JVT 所采纳以取代原先的 UVLC（Universal Variable Length Coding）[25,26]。由于它被用来编码一些重要的参数，所以优先考虑的是纠错能力而非编码效率。

Exp-Golomb 码的编码过程分为两步：第一步将待编码的数据转换为一个中间变量 codeNum。根据转换方式，可将 Exp-Golomb 码分为 4 种，按 H.264 的记号分别表示为 ue, me, se 和 te。其中 ue 和 se 分别用于无符号整数和有符号整数的编码，me 用于对 coded_block_pattern 编码，这是表示当前宏块中哪些子块含有非零系数的一个参数，te 用于编码 ref_idx_l0 和 ref_idx_l1，该参数表示使用缓存中的哪个参考帧做帧间预测。第二步是将 codeNum 映射为二进制编码。具体编码过程很简单，这里不再赘述。

2.4.2 CAVLC

CAVLC 是基于上下文的自适应变长码的英文简称，它用于编码预测残差。VLC 的基本思想是对经常出现的符号赋予较短的码字，反之则较长。与一般的 VLC 码不同的是，CAVLC 能够根据以往编码的数据在若干码表中自适应地选择，找出与当前编码数据统计特性最相符的一个码表来进行编码。并且它将以前标准中所采用的(Run, Level)二元组拆开来，分别进行编码，从而能达到更好的自适应性，提高了编码性能[27, 28]。由于 CAVLC 改进的编码性能以及实现简单，它被 JVT 所采纳，同上面的 Exp-Golomb 码一起代替了原先的 UVLC。

CAVLC 的设计考虑了如下几个事实：1）经过变换与量化后的预测残差中含有较多的 0，这样在 Zig-Zag 扫描之后，用 Run 和 Level 表示预测残差可以取得较好压缩效果。这一点在以前的标准中也用到了。这里 Level 表示非零系数值，

Run 表示非零系数之前的 0 的数目。2）残差末尾的几个非零系数一般为±1（trailing 1s, T1s），CAVLC 对它们单独进行了编码。3）作为空间相关性的一种表现，当前块中的非零系数数和周围块的中的非零系数数应该差不多，CAVLC 利用这一点自适应地选择编码当前块中非零系数数的码表。4）位于低频处的系数值一般较大，而位于高频处的则相反，CAVLC 利用这一点自动地选择编码 Level 的码表。5）位于低频处的非零系数一般是连着的，中间没有零，此时用(Run, Level)形式来表示它们就显得效率不高，因此 CAVLC 将 Run 和 Level 分开单独进行编码。

图 2-12 是用 CAVLC 编码的基本流程。

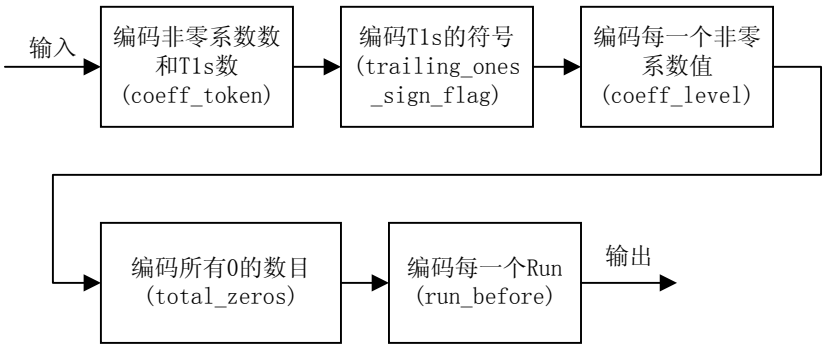


图 2-12 CAVLC 编码流程图

第一个被编码的语法元素（syntax element）是 coeff_token，它包含两个信息：所有的非零元素个数（total_coeff，包括 T1s）和 T1s 的个数。total_coeff 可取 0 到 16，T1s 可取 0 到 3。一共有 5 个码表供编码器进行选择：除了专用于 chroma DC 系数的码表之外，其余 4 个码表分别对不同的 total_coeff 取值范围做了优化，表 0 适用于 total_coeff 较小的情况，对比较小的 total_coeff 赋予较短的码字，表 1 则适用于 total_coeff 取值中等的情况，对中等取值的 total_coeff 赋予较短的码字，依此类推。如前所述，由于空间相关性，当前块中的 total_coeff 与周围块的十分相近，因此可以依据周围块的 total_coeff 从 4 个码表中选择一个最好的表进行编码，这就体现了 CAVLC 的上下文自适应的能力。

随后被编码的是 T1s 的符号，0 表示正，1 表示负。CAVLC 对残差数据的编码顺序与 Zig-Zag 扫描的顺序相反，即从高频向低频走。

第三个被编码的是 T1s 之后的 Level 数据。对它的编码也体现了 CAVLC 的上下文自适应的能力。同 coeff_token 类似，CAVLC 根据 Level 的不同取值范围

建了 7 个码表 VLC0-VLC6，VLC0 倾向于给较小的 Level 赋予较短的码字，而 VLC1 会给稍大一点的 Level 赋予较短的码字，依此类推。随着 Level 值由小到大地变化，CAVLC 会根据先前被编码的 Level 决定是否地转至下一个码表，判断依据就是给每个码表设一个阈值，如果大于则跳转。表 2-6 给出了 VLC0-VLC6 的阈值。

表 2-6 编码 Level 时 VLC0-VLC6 的阈值

VLC	阈值
VLC0	0
VLC1	3
VLC2	6
VLC3	12
VLC4	24
VLC5	48
VLC6	∞

初始码表是 VLC0，但如果 total_coeff>10 且 T1s<3 的话则选 VLC1。每编码一个 Level，编码器便将其与表 2-6 中的阈值进行比较，如果 Level>阈值，则转至下一个码表，从表中可以看出，VLC0 最多被使用 1 次，因为它的阈值为 0。这里需要说明一个特殊情况，当 T1s<3 时，第一个 Level 肯定不会取±1，所以对这个 Level 编码时将其绝对值减 1 以减小码长。

各个 VLC 的结构和内容在比较早版本的标准中有所描述[29]，但最近的版本却将其取消了，代之以一种指令性的文字性描述。我认为如果仅仅是为了节省篇幅，这样做似乎没有必要。一来这段晦涩的文字隐藏了码表选择的原理，二来作为标准，定义的应该是码流的含义与结构，而不应该出现类似程序代码的指令性文字，因为如何进行码表的选择和译码是开发者的事，标准不应对如何实现做出具体的规定，即使这种实现方式可能是最优的。关于 VLC0-VLC6 的具体结构和内容请参见[29]，为方便读者，这里列出 VLCN, $1 \le N \le 6$ 中码字构成的公式：

if (|Level|-1) < (15<<(N-1)),

Code: 0….01x..xs,

where number of 0's = (|Level|-1) >> (N -1),

number of x's = N-1,

value of x's = (|Level|-1) % $2^{(N-1)}$,

（这实际上相当于取 Level 的低 N-1 位）

s = sign bit (0 – positive, 1 – negative)

else,

28-bit escape code: 0000 0000 0000 0001 xxxx xxxx xxxs,

where value of x's = (|Level|-1) - (15<<(N -1)),

s 的含义同上

CAVLC 的第四步是编码最后一个非零系数之前所有 0 的个数 total_zeros。以往的标准中，每一个非零系数都与一个二元组(Run, Level)相联系，编码是针对这个二元组进行的。前面讲过，残差开头的几个非零系数一般是连着的，它们之间没有 0，因此它们的 Run 就没必要编码。因此 CAVLC 就将二元组(Run, Level)拆开，Run 与 Level 分别编码，这样就能够提高压缩效率。显然有 total_zeros≤total_coeff，所以 total_coeff 可以决定 total_zeros 的取值范围，并且通过观察可以发现，不同的 total_coeff 下 total_zeros 的分布也有不同，所以 CAVLC 根据不同的 total_coeff，建立了不同的码表用于对 total_zeros 的编码，这也体现了 context adaptive。

最后一步就是对 Run 进行编码。CAVLC 引入了一个变量 zerosLeft，表示还没有编码的 0 的个数，初值为 total_zeros。显然 Run 的取值范围必须在 0 到 zerosLeft 之间，这样 zerosLeft 就决定了可以用多少 bit 来表示 Run。例如，如果 zerosLeft=1，那么 Run 只能取 0 或 1，所以只需 1 bit 就可以对其进行编码，而如果 zerosLeft=6，那么就需要 3 bit 因为 Run 的取值范围在 0 到 7 之间。当然，这说的是定长码的情况，CAVLC 根据不同 zerosLeft 下 Run 的分布建立了不同的 VLC 码表，供编码器选择。这也体现了 context adaptive 的特点。

当 Run=zerosLeft 时，编码器就可以停止编码，哪怕这个 Run 对应的 Level 不是第 1 个非零系数。因为剩下的非零系数（如果有的话）前面都没有 0 了，不需要再进行编码了。这就体现出 Run 和 Level 分开编码的优越性。下面我举个例子说明一下 CAVLC 的编解码过程。

假设经过 Zig-Zag 扫描后的 luma 系数如下所示

-5, 4, -2, 0, 2, 0, 0, 1, -1, 0, …

则 total_coeff=6, T1s=2, total_zeros=3，令 zerosLeft=total_zeros=3

表 2-7 CAVLC 举例——编码过程

编码项目	输入数据	编码结果	描述
coeff_token	total_coeff=6, T1s=2	001101	①
T1s 符号	-	1	

	+	0	
Level	2	1	②
	-2	0011	③
	4	00010	④
	-5	00101	⑤
total_zeros	total_coeff=6, total_zeros=3	110	
Run	Level=-1 zerosLeft=3, run_before=0	11	⑥
	Level=1, zerosLeft=3, run_before=2	01	⑦
	Level=2, zerosLeft=1, run_before=1		⑧

① 假设根据周围块的 total_coeff 算出采用对应 4<=nC<8 的 VLC 表

② 初始化采用 VLC0，由于 T1s<3，所以绝对值减 1，按 1 编码

③ 2>0，转至 VLC1

④ |-2|<3，仍用 VLC1

⑤ 4>3，转至 VLC2

⑥ 编码后 zerosLeft 减 0

⑦ 编码后 zerosLeft 减 2

⑧ zerosLeft=run_before，编码停止

最终的编码结果是（从左向右）：

00110110100110001000101110110 1

在编到 2 的 Run 时，编码器就停止了，开头 4 个系数的 Run 不必再编码，由此可以看出通过引入 total_zeros 实现 Run Level 分开编码所带来的好处。

对其解码的过程如下表所示：

表 2-8 CAVLC 举例——解码过程

解码项目	输入码	解码结果	输出数据	描述
coeff_token	001101	total_coeff=6, T1s=2		①
T1s 符号	1	-	-1	
	0	+	1, -1	
Level	1	2	2, 1, -1	②
	0011	-2	-2, 2, 1, -1	③

	00010	4	4, -2, 2, 1, -1	④
	00101	-5	-5, 4, -2, 2, 1, -1	⑤
total_zeros	110	total_zeros=3	-5, 4, -2, 2, 1, -1	⑥
Run	11	run_before=0	-5, 4, -2, 2, 1, -1	⑦
	01	run_before=2	-5, 4, -2, 2, 0, 0, 1, -1	⑧
			-5, 4, -2, 0, 2, 0, 0, 1, -1	⑨

① 在解码端根据周围块的 total_coeff 算出的 VLC 码表号与编码端相同

② 初始化采用 VLC0，由于 T1s<3，所以解码后绝对值加 1

③ 2>0，转至 VLC1

④ |-2|<3，仍用 VLC1

⑤ 4>3，转至 VLC2

⑥ 解码根据 total_coeff=6, 解码后令 zerosLeft=3

⑦ 解码根据 zerosLeft=3，解码后 zerosLeft 减 0

⑧ 解码根据 zerosLeft=3，解码后在 1 前面添 2 个 0，zerosLeft 减 2

⑨ zerosLeft=1，所剩的一个 0 添在 2 前面，

最终的解码结果是（从左向右）：

-5, 4, -2, 0, 2, 0, 0, 1, -1, …

2.4.3 CABAC

由于时间有限，未能对 CABAC 做深入的研究，许多地方只知其然而不知其所以然，所以如有不当之处请读者指正是幸。

前已提及，算术编码是一种高效的熵编码，从整体上看，它能够使每一个符号获得分数个数的 bit，从而最大限度地逼近信源的极限熵。虽然算术编码最初的定义在概念上很简单，可实现起来却很复杂，因为随着编码符号个数的增加，它需要无穷精度的小数来表示编码结果，并且编码过程中也需要大量的小数运算，如子区间的划分。另外它需要信源统计特性的先验知识，不能够根据以前已经编过码的符号估计信源的概率分布，没有自适应能力。为了提高编码效率并解决复杂性和自适应性的问题，H.264 引入了 CABAC[30-32]，它除了具有算术编码固有的优点之外，还有以下特点：

（一）具有自适应能力。CABAC 在编码过程中能够动态地估计符号的概率，下一个待编码的符号将按新的概率进行编码。

（二）传统的自适应算术编码（AAC）在编码的过程中累计接收到的符号数目，用某个符号数目在总的符号数目中所占的百分比作为对其概率的估计，在实现时需要用到除法运算。CABAC 的自适应原理与之相同，但是采用了有限状态机（finite state machine, FSM）[32]与查表来实现对概率的估计。FSM 的每一个状态对应一个典型概率，这实际上是对概率进行了量化。估计符号概率的过程相当于在不同的状态间切换。这样建立一个状态转换表就可以用简单的查表来完成概率更新，从而避免了传统 AAC 中概率估计所用的除法运算。

（三）CABAC 同样用查表的方法完成子区间的划分。一般地，如果想得到子区间的长度，需要做乘法 Range*p，其中 Range 代表当前区间的长度，p 为概率。为避免乘法运算，CABAC 将 Range 进行了量化，由于概率 p 已经被离散地表示成 FSM 的一个个状态，所以 Range*p 的结果就是有限个数，可以事先计算好作成一张表，这样以当前区间的量化值和状态作为索引就可以查表得到子区间的长度了。

总之，CABAC 可以获得比 CAVLC 更好的自适应性和编码性能，其代价就是相对较高的复杂性。

图 2-13 是 CABAC 编码的总体流程：

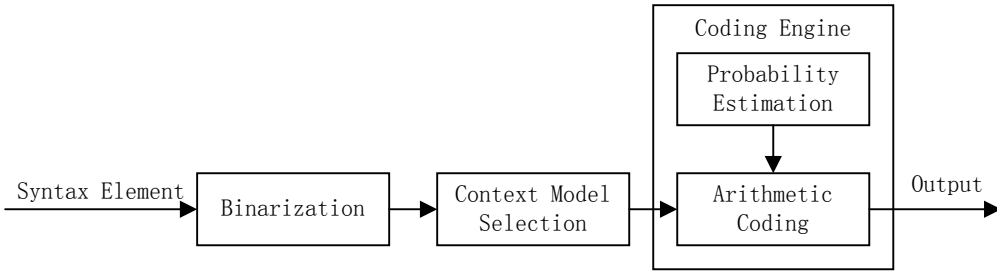


图 2-13 CABAC 编码的总体流程

CA**B**AC 只对二进制符号“0”和“1”进行编码，所以首先要对输入的 Syntax Element（SE）的值做二值化（Binarization），即映射成一串 01 序列，序列中的每一个二进制符号称为一个“bin”，bin 是 CABAC 的编码单元。对每一个 bin 的编码包括两个步骤：选择 context 和进行算术编码。

由于空间相关性，SE 的取值与邻近相同的 SE 取值是相关的，这样，描述当前 SE 的概率就应该用条件概率模型。Context 就是这样一个条件概率模型，每一

个 context 描述了在邻近 SE 的某一个取值或取值组合下，当前 SE 中某一个 bin 的条件概率。由于一般邻近 SE 有多种可能取值，所以当前 bin 也对应有多种不同的 context（以 ctxIdx 索引），这就需要编码器根据邻近 SE 的取值选取不同的 context（也就是 ctxIdx）对当前 bin 进行编码。这里邻近的含义不仅包括空间上相邻，而且也包括先前编码的 bin，即时间上相邻。

在选择了合适的 context 之后，编码器就开始算术编码，在编码完成后编码器更新 bin 的概率（实际上是 FSM 的状态），记录在所用的 context 中。如前所述，编码过程中计算子区间长度和概率估计均采用查表操作，避免了乘除法运算，而其它的一些运算和操作也是针对整数进行的。

当前 bin 的编码完成后，编码器对下一个 bin 重复上述两个步骤直至当前 SE 的所有 bin 都编码完毕。

实际的 CABAC 在实现过程中还有很多技术细节，如怎样二值化，如何由某个具体的 bin 确定 ctxIdx，这在标准中都有详尽的叙述。标准中这些数据是根据大量的训练集产生的，由于本人时间有限，未能重复这一训练过程。

第三章　H.264 的软件实现

本章介绍 H.264 的软件实现。由于时间所限，我仅实现了编码器，且重点在帧内编码。本软件参考了文献[4]。

3.1 开发环境

本软件所用的开发工具是 Borland C++ Builder 6，之所以选择它而非 VC，一方面是由于其出色的界面设计能力，另一方面是本人对 Borland 公司的开发工具较为偏爱，再一方面是文献[4]已经在 VC 环境下编译过，为避免抄袭嫌疑，故采用了另一开发系统，事实上，在 BCB6 中需要对[4]进行较大的修改才能编译通过。本软件所用的操作系统是 Microsoft Windows 98，运行平台是一般台式 PC 机，其中对运行速度有较大影响的参数为：Intel 赛扬 850 CPU，128M PC133 SDRAM 内存，硬盘转速为 7200。

3.2 软件介绍

本软件的功能是对视频序列进行 H.264 编解码。需要注意的是，本软件仅用于作者本人研究 H.264 视频编码标准，并非通用商业软件，因此并不提供细致的参数检查，所以在使用时应按本文档要求输入合理的参数，否则软件将可能不会正常工作。此外软件中可能含有第三方专利，请注意尊重相关知识产权，作者保留一切权利。详细信息请单击程序中的“关于”按钮查看。

软件的输入是 YUV 格式的文件，输出为编码后的文件和解码后的重建图像文件（也是 YUV 格式）。执行过程分为四步：第一步是指定输入输出的文件名；第二步是设置编码参数；第三步进行编码；第四步显示编码结果。

为方便操作，软件界面采用 Wizard 形式，用户只需用鼠标单击“下一步”或“上一步”按钮即可完成操作。

启动软件后将显示如图 3-1 所示的第一步的界面。此步骤用于指定输入文件以及作为输出的编码文件和重建文件。默认情况下，输入文件和重建文件均为 YUV 格式，编码文件后缀为.264。用户可以在输入框中直接键入，也可以单击

输入框右端的图标，然后在打开文件对话框中选择。如果利用打开文件对话框选择了输入文件而某个输出文件尚未指定，程序将自动对其指定——对于编码文件其文件名与输入文件相同而后缀为.264；对于重建文件将在文件名后添加“_rec”而后缀为.YUV。

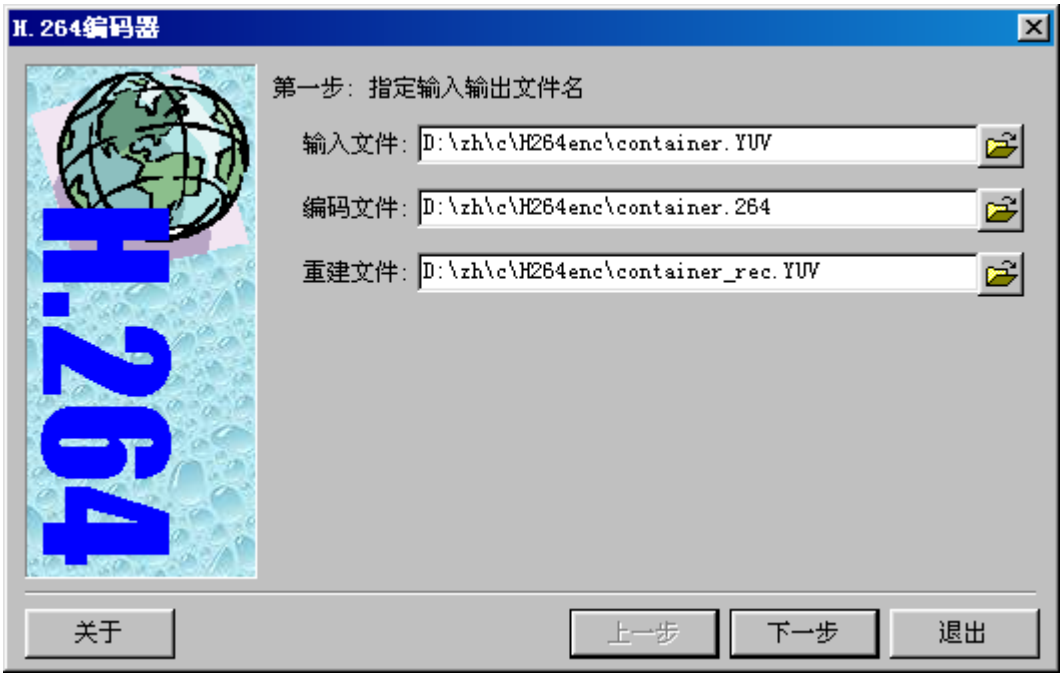


图 3-1 第一步界面：指定输入输出文件

指定了输入输出文件后，单击“下一步”按钮将转入如图 3-2 所示的第二步界面。注意第一步的信息必须全部输入，否则程序将要求用户输入而不转入第二步。

第二步是设置编码所用的一些参数，其中待编码的参考帧（anchor frame）数目指的是 I 帧与 P 帧的数目，而非所有要进行编码的帧的数目。为简化程序，规定在显示顺序上最后编码的帧必须为参考帧，这样所有要进行编码的帧数就是参考帧的数目加上参考帧间 B 帧的数目，显然，这个输入框的数据必须大于 0。下面两个输入框用于指定 GOP 结构，其含义很明显，这里不多做解释，需要说明的是，I 帧在参考帧中的间隔周期一栏如果输入的是 0，表示只对视频序列的第一帧按 I 帧编码，随后的参考帧全部按 P 帧处理。如果上面两栏的输入数据正确的话，GOP 结构将在“GOP 结构”栏中显示出来以供参考。

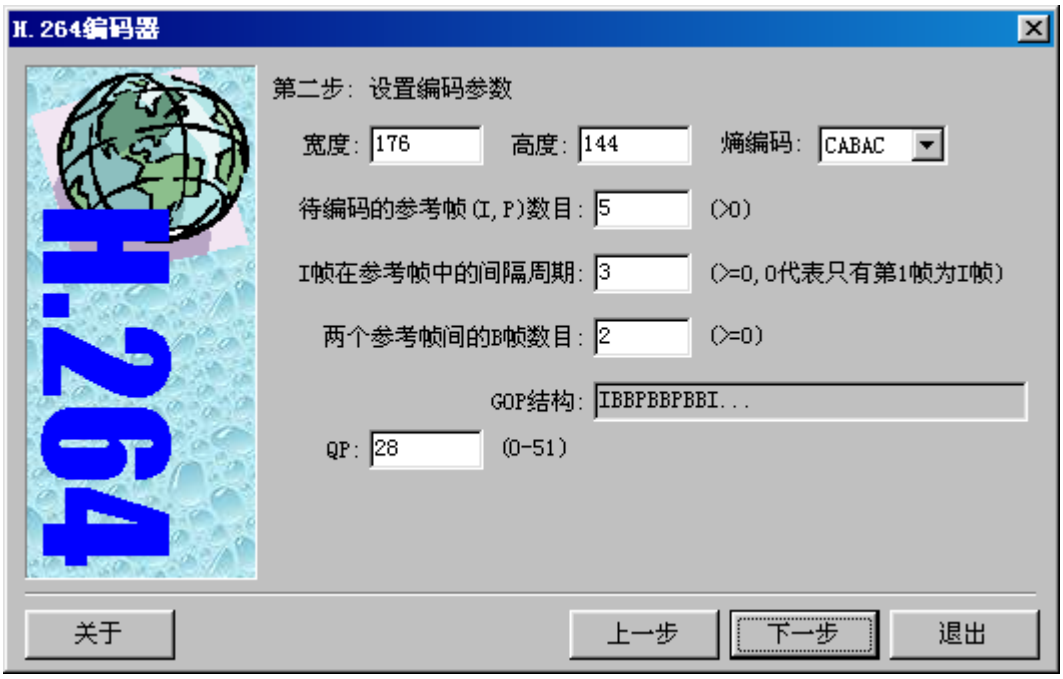


图 3-2 第二步界面：设置编码参数

此时单击“上一步”按钮，程序将回到第一步的界面，用户之前输入的数据仍然保留。如果所有输入的编码参数都正确，单击“下一步”按钮，程序就将按照这些参数开始进行编码，如图 3-3 所示。

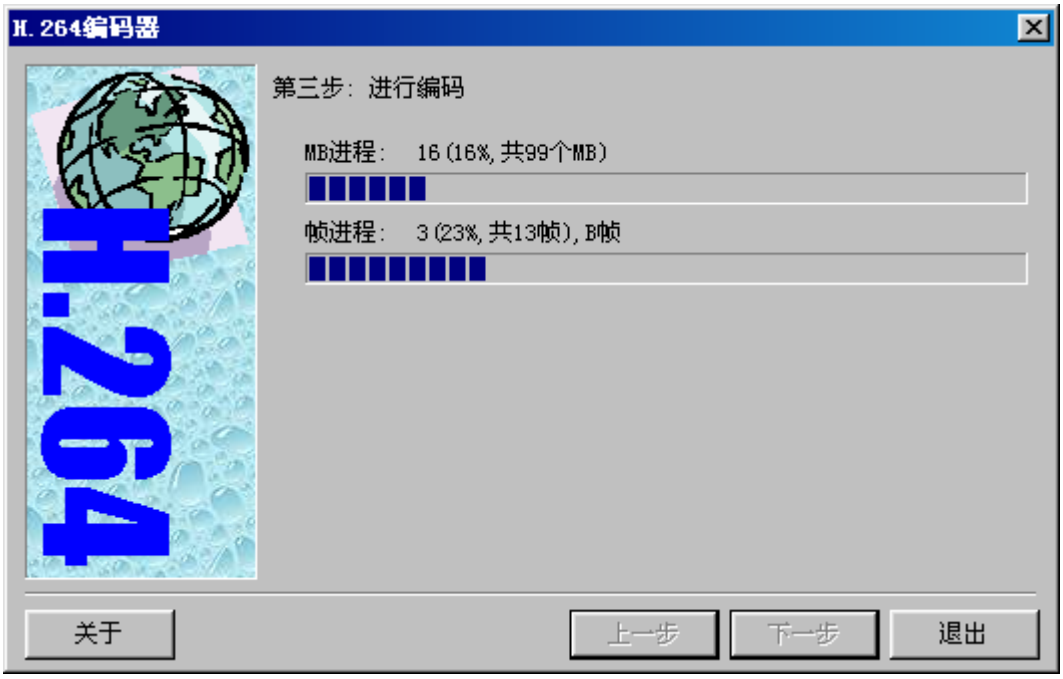


图 3-3 第三步界面：进行编码

第三步的任务是完成对输入文件的编码。程序用进度条显示了当前编码的进度，包括帧进度和当前帧内的宏块（MB）进度。同时显示的信息还包括当前帧

的序号（从 1 开始）、百分比、总共待编码的帧数和当前帧的类型，以及当前帧所含的 MB 数、当前 MB 的序号和百分比。这一步程序使用了两个并行执行的线程（Thread），一个线程在后台进行编码，一个线程在前台与用户交互并实时刷新输出。

编码完成后程序转入第四步界面，如图 3-4 所示。这时窗口中列出了刚才编码的一些统计信息，包括各帧的统计信息和总的统计数据，供用户参考。此时单击“上一步”按钮可以回到第二步，单击“重新开始”按钮将从第一步开始重复上述的操作过程。

图 3-4 第四步界面：显示编码结果

3.3 编码结果分析

表 3-1 所示的是对 VCEG 的标准测试序列[34]container 进行编码的输出结果，编码参数如图 3-2 所示。

表 3-1 一个编码结果

```
-------------------------------------------------------
输入文件：D:\zh\c\H264enc\container.YUV
编码文件：D:\zh\c\H264enc\container.264
重建文件：D:\zh\c\H264enc\container_rec.YUV
----------------------各帧统计信息------------------------
 帧     Bit 数    SNR-Y     SNR-U     SNR-V    时间(ms) Intra_MB
```

0(I)	24184	37.4246	41.3328	40.8001	1820	
3(P)	1104	36.9877	41.3584	40.7300	3350	0
1(B)	112	37.2180	41.3722	40.7358	4840	
2(B)	192	37.0615	41.2663	40.7397	8120	
6(P)	1360	36.7150	41.3814	40.7312	4340	0
4(B)	96	36.8368	41.3717	40.7226	7140	
5(B)	112	36.6519	41.3734	40.7170	5060	
9(I)	24280	37.2439	41.2499	41.1443	1920	
7(B)	160	37.6478	41.7077	41.3078	4730	
8(B)	96	37.6668	41.6778	41.3154	4610	
12(P)	1168	36.8654	41.4142	41.1526	3300	0
10(B)	128	37.1216	41.2944	41.1353	4610	
11(B)	112	36.9049	41.3676	41.1278	4660	

共编码帧数 ： 13 (I:2, P:3, B:8)
共用时间 ： 58.500 秒
图像大小 ： 176x144
熵编码方法 ： CABAC
QP ： 28

----------平均 SNR 统计数据----------

	SNR-Y	SNR-U	SNR-V
I 帧	37.3342	41.2914	40.9722
P 帧	36.8561	41.3847	40.8713
B 帧	37.1387	41.4289	40.9752
所有帧	37.1035	41.3975	40.9507

----------平均 bit 数统计数据----------

	总共 bit 数	平均每帧 bit 数	平均码率(bps)
I 帧	48464	24232	726960
P 帧	3632	1211	36320
B 帧	1008	126	3780
所有帧	53104	4085	122548

----------其它统计数据----------

Intra 4x4 MB 数 ： 115
Intra 16x16 MB 数 ： 83
用于 Parameter Set 的 bit 数 ： 160

表 3-1 中的 SNR-Y, SNR-U, SNR-V 分别表示 Y, U, V 分量的 PSNR（peak signal-to-noise ration）。PSNR 的定义如下[12, 33]：

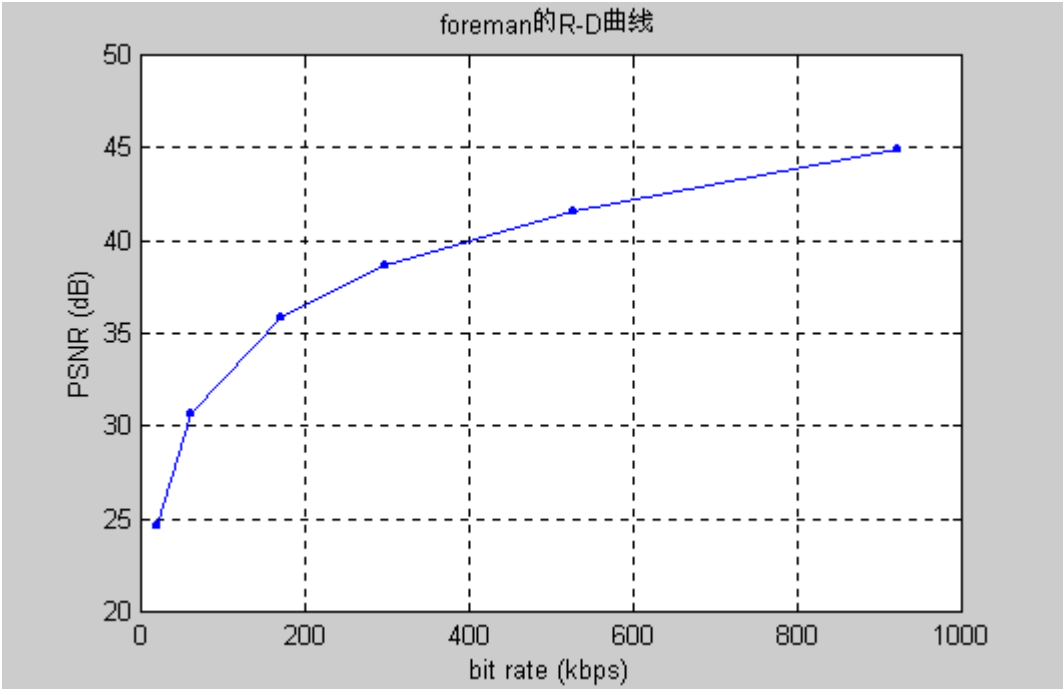
$$\mathrm{SSD} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left|f(x,y)-g(x,y)\right|^2 \tag{2.1}$$

$$\mathrm{MSE} = \frac{1}{MN}\mathrm{SSD} \tag{2.2}$$

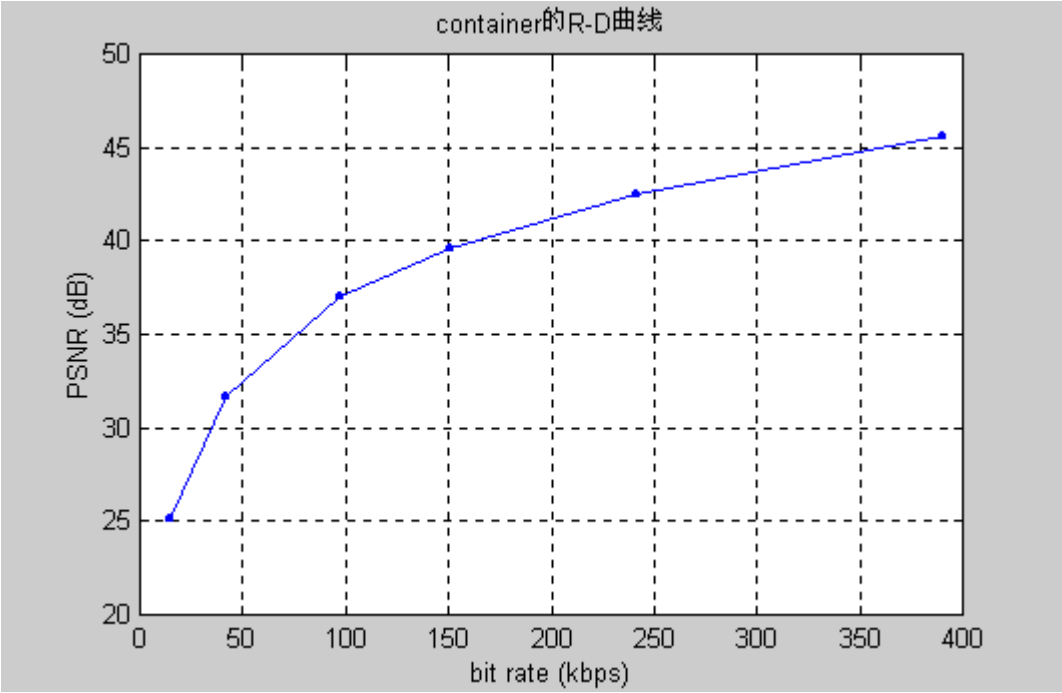
$$\mathrm{PSNR} = 10\lg\left(\frac{255^2}{\mathrm{MSE}}\right) \tag{2.3}$$

其中 $f(x,y)$ 表示未经编码的原始输入图像， $g(x,y)$ 是重建图像，M, N 分别表示图像的宽和高，PSNR 的单位是 dB。由定义知，PSNR 越大，失真越小，图像的质量越高，反之亦然。Intra_MB 表示 P 帧中采用帧内编码的宏块个数。由于表 3-1 中的视频序列运动较缓慢，所以没必要用帧内编码，但对于运动较剧烈的视频序列，如 foreman，为保证图像质量，该列项目就不为 0 了。

R-D 曲线可以比较全面地表示编码性能。图 3-5 绘出了两个[34]推荐的参考视频序列：foreman 和 container 的 R-D 曲线，foreman 的特点是运动比较剧烈且摄像头在不停地晃动，container 的特点是图像运动比较缓慢且背景固定。它们的图像格式均为 QCIF（176x144），4:2:0，30 帧/秒。编码所用的 GOP 结构为 IBBPBBPBBI…，熵编码采用 CABAC，曲线上各点分别对应 QP=16, 20, 24, 28, 36, 46。按[34]要求，图中表示 D 的纵轴是 luma 分量 PSNR 的平均值，单位为 dB，而表示 R 的横轴是平均比特率，单位为 kbit/s。这里的平均值是对序列中所有帧取的，其中 foreman 为 400 帧，container 为 300 帧。需要注意的是，RD 曲线应该是包含各点的一个凸闭包（convex hull），这里为了作图方便，仅用直线将各点相连。



(a)



(b)

图 3-5 R-D 曲线，QP=16, 20, 24, 28, 36, 46

(a)foreman，对 400 帧取平均 (b)container，对 300 帧取平均

图 3-5 所绘的曲线反映了在某一种特定的编码模式下编码器的性能，当 QP 变化时，对应的(R, D)也随之变化，从而形成一条 R-D 曲线，不同的编码模式对应不同的 R-D 曲线。信息论告诉我们，对于某个信源，都有一个理论上的 R-D 界，任何编码方法都不可能超出这个界，因此，一个好的编码器其 R-D 曲线应该最大限度地逼近这个理论上的 R-D 界。这也说明了评价一个编码器的好坏不能笼统地说它的码率低或失真小，而应看它整体的 R-D 性能。

这里举一个例子。对于序列 container，由于它运动比较缓慢，所以多采用 P 帧或 B 帧编码可以获得比较好的性能，这一点可以从相对应的不同 R-D 曲线上反映出来。下面我们用不同的 GOP 结构来对 container 进行编码，以比较它们各自的 R-D 性能。这里用 I 表示 I 帧在参考帧中的间隔周期，用 B 表示在两个参考帧中 B 帧的数目。第一种 GOP 就是图 3-5 中所用到的 I=3, B=2; 第二种：I=5, B=2，第三种：I=5, B=4。它们的 R-D 曲线绘于图 3-6，由于在低比特率处曲线相距较近，为便于观察，横轴采用对数坐标。作为比较，表 3-2 列出了不同 QP 下三种编码模式所得到的 R, D 值。

从图表中可以看出，减少 I 帧数目或增加 B 帧数目都可以使编码性能改善，这是因为图像运动缓慢，P 帧或 B 帧的 MV 和预测误差均较小，所以多使用帧间

预测编码可以在 D 下降很小的情况下大幅地降低码率，这一点在低比特率处尤为明显。因此我们说，通过改变 GOP 结构，可以使编码器的性能在 R-D 意义上得到改善，也就是说在相同的码率下可以得到更佳的图像质量；在相同的图像质量下所需的码率更小。但是，对于运动较剧烈的图像，如 foreman，这样的措施便不能够取得很好的效果，如果绘出它们的 R-D 曲线，可以想见，相互之间一定靠得很紧。

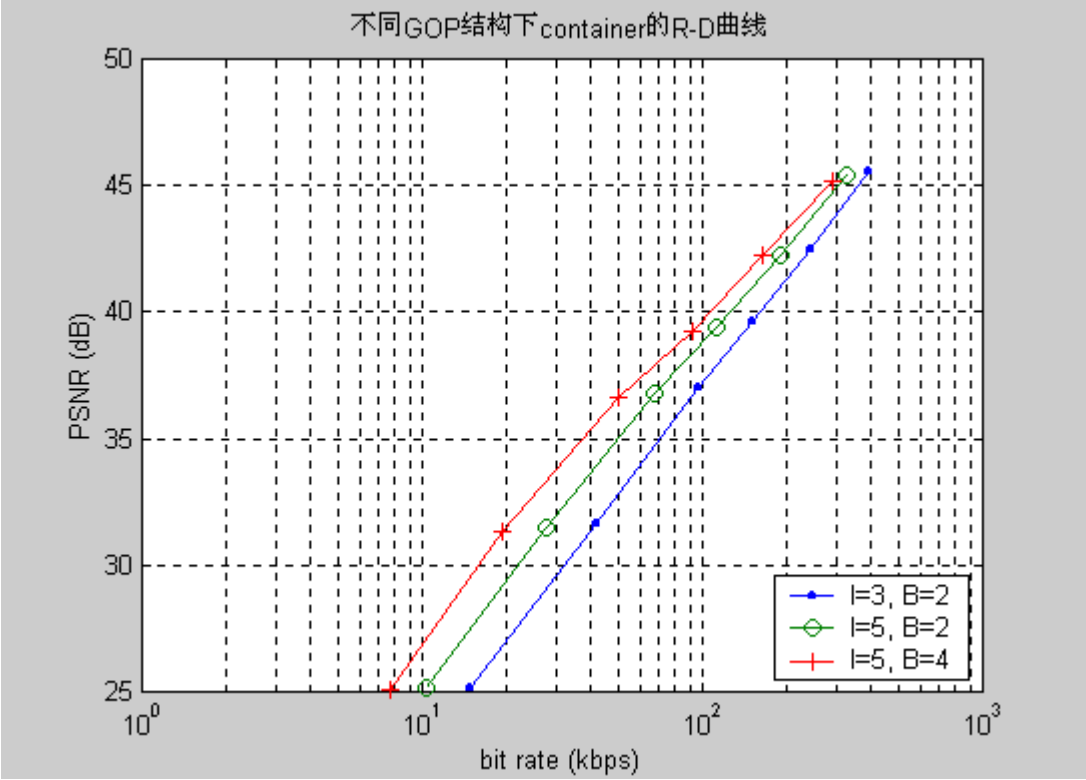


图 3-6 不同 GOP 结构下 container 的 R-D 曲线

表 3-2 三种编码模式下 RD 值的比较。从表中可以看出，右侧的模式与左侧的相比，在 PSRN 降低很小的情况下，码率有了较大的下降

QP	luma PSNR (dB)			bit rate (kbps)		
	I=3, B=2	I=5, B=2	I=5, B=4	I=3, B=2	I=5, B=2	I=5, B=4
16	45.6088	45.4235	45.1916	390.553	327.434	289.828
20	42.4919	42.2800	42.2383	241.530	190.762	163.361
24	39.6258	39.3687	39.2757	150.528	111.645	91.500
28	37.0021	36.7655	36.6165	97.068	67.491	50.526
36	31.6593	31.4880	31.3564	42.049	27.668	19.218
46	25.1890	25.1232	25.0709	14.748	10.279	7.648

3.4 存在问题及解决方法

这个软件最大的问题就是实时性差。例如，为了获得图 3-5(a)中的一个点，需要大约半个小时的时间，而 400 帧图像在 30fps 的速率下播放不会超过 14 秒。从这可以看出编码所消耗的时间是惊人的，这极大地制约了 H.264 的实用化。对于这种问题解决的途径可以分成两个层次：第一个层次是在所运行的平台上优化代码，比如对于使用 Intel CPU 的 PC 平台，可以在关键代码处采用 MMX、SSE 或 SSE2 指令集，另外也可以用 equator 公司的 MAP-CA 等具有 VLIW 特点的高档 DSP 来实现。前面已经提到过，H.264 提出的背景是硬件技术，特别是半导体芯片技术的飞速发展，仅以 Intel 的 CPU 为例，在我刚刚考上研究生的时候，主流产品还是 500M 的 PII，可现在 1.6G 的 P4 早已大行其道，所以我完全有理由相信，三年以后的硬件技术绝对可以给 H.264 以强大的支持。

当然，仅仅依靠这一层次的优化是不够的，我们还应当进入第二个层次：开发快速算法。分析这个软件的实现算法，可以发现造成其性能瓶颈的因素主要有两个，一个是模式选择（mode decision），再一个是全搜索（full search）运动估计（ME）算法。对于前者，由于采用了一种穷举式的模式选择算法，导致帧内编码时间很长，我们可以利用预测模式的方向性开发出一种快速算法，这在下一章将详细叙述。而对于后者，虽然全搜索可以找到全局最优的 MV，但它在搜索范围内的每一点都测量 SAD，再加上 H.264 对 luma 是 1/4 搜索精度，对 chroma 是 1/8 精度，所以很浪费时间。目前已经有很多快速 ME 算法[37-39]，它们都能够在不明显降低预测精度的前提下提高 ME 速度，由于我在这方面研究的不多，所以就不深入介绍了。这里我想提一下，随着量子计算机研究的开展，一个处理单元可以仅用几个原子实现，这样一块芯片中可以有成千上万个并行运行的编解码模块，那时穷举算法将可能成为最好的选择，模式选择和全搜索也许就不再成为令人头疼的问题了。

第四章　帧内预测模式选择的快速算法

本章介绍了一种基于图像边缘方向检测的快速帧内预测模式选择算法，实验表明，该算法能在编码性能下降很小的前提下，极大地缩短确定帧内预测模式的时间。

4.1 问题的提出和解决思路

前面介绍过，H.264 提供了许多种编码模式，例如仅帧内编码的 luma 分量就有 4 种 Intra_16x16 模式和 9 种 Intra_4x4 模式，再加上 4 种 chroma 分量的模式，这样一个 MB 可能的模式组合就有(16*9+4)*4=592 种。而该软件在实现时为了找到 R-D 意义上最优的 mode，采用了基于拉格朗日乘子法的率失真优化（RDO）算法，即最小的 $J=D+\lambda R$ 所对应的 mode 被认为是最优的。这样对每一个 MB，程序要计算 592 次 $J=D+\lambda R$，而 D 和 R 都要经过实际的编码和解码过程才能得到，因此每编码一个 MB 相当于做了 592 次编码和解码。尽管这里的解码不需要熵解码过程，有些 MB 也不是所有的模式组合都可用，但这种穷举算法显然太耗时。如果在 P 帧中再和 7 种帧间预测模式组合一下，以判断采用帧间编码还是帧内编码，那编码一个 MB 所用的时间几乎和实时编码一帧的时间一样，这是导致程序运行缓慢的主要原因。

为了解决这个问题，让我们回想一下在 2.2 节介绍帧内预测的特点时，曾经提到过除了 DC 模式，其它所有模式在预测时都是具有方向性的，即沿某个预测方向“复制”（外插）参考像素的灰度值来形成预测值。如果当前块的纹理方向，或边缘方向恰好和预测方向一致，则预测就能够相当准确，由此得到的预测残差也就很小，可以获得相当大的压缩率。这一事实就是下面介绍的基于图像边缘方向检测的快速帧内预测模式选择算法的思路和原理：即首先检测出当前块的边缘方向，然后找出一个最接近的预测方向，所对应的预测模式就认为是最佳模式，如果有多个模式候选，再利用 RDO 确定出一个 R-D 最优的预测模式。

这一思路部分来源于文献[40]，但作者做了一定的改进，表现在：1）利用边缘方向角的正切判断预测方向，2）对 DC 的处理有所不同，3）对 Intra 4x4

模式采用在直方图上取值最大的三个值作为候选，4）直方图统计的是像素个数而非ΣAmp。

4.2 算法的推导

4.2.1 像素边缘方向的检测

首先定义边缘方向角的概念，如图 4-1(a)所示。为直观起见，图中使用了一个灰度值渐变的背景表示图像的边缘状况，点 O 是图像中的某一点，从图中可以看出，其边缘方向是从左下方至右上方。定义边缘方向与水平方向的夹角 θ 为边缘方向角，简称方向角，且取值范围为 $(-90°,90°]$。类似地也可以定义预测方向与水平方向的夹角为预测角度，且其取值范围亦为 $(-90°,90°]$，不同模式的预测角度请参见表 4-1。

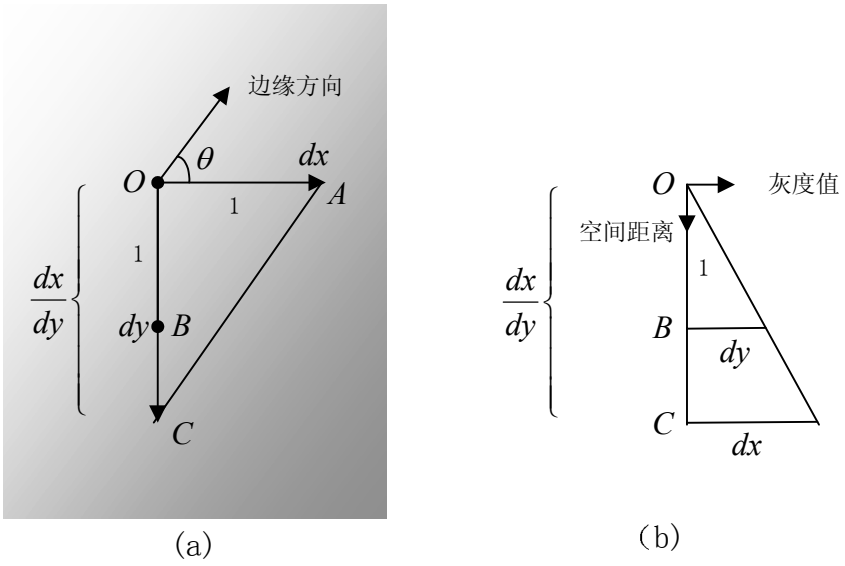


图 4-1 (a)边缘方向角 (b)点 C 离点 O 的距离

图中，假设 dx 为 O 点沿水平方向在单位距离上灰度值的增量，dy 为垂直方向单位距离上的增量，并假设平行于边缘方向的线段 AC 上的点灰度值都相同，由图 4-1(b)容易算出，点 C 离点 O 的距离为 $\frac{dx}{dy}$，则方向角可以用 dx 和 dy 表示为：

$$\theta = \arctan\left(\frac{dx}{dy}\right) \tag{4.1}$$

dx 和 *dy* 的值可以通过 Sobel 边缘检测算子求得[41]，其水平和垂直方向上的 Sobel 算子分别如图 4-2(a)和(b)所示：

−1	0	1
−2	0	2
−1	0	1

−1	−2	−1
0	0	0
1	2	1

图 4-2 Sobel 算子

(a)水平 Sobel 算子，用于计算 *dx* (b)垂直 Sobel 算子，用于计算 *dy*

用公式表示就是：

$$dx_{i,j} = p_{i+1,j-1} + 2p_{i+1,j} + p_{i+1,j+1} - p_{i-1,j-1} - 2p_{i-1,j} - p_{i-1,j+1} \tag{4.2}$$

$$dy_{i,j} = p_{i-1,j+1} + 2p_{i,j+1} + p_{i+1,j+1} - p_{i-1,j-1} - 2p_{i,j-1} - p_{i+1,j-1} \tag{4.3}$$

其中 *i* 和 *j* 表示横坐标和纵坐标，正方向分别为向右和向下，$p_{i,j}$ 为位于坐标 (i,j) 处的像素值，$dx_{i,j}$ 和 $dy_{i,j}$ 分别表示与坐标为 (i,j) 的像素相联系的 *dx* 和 *dy* 值，该像素对应图 4-1 中的点 *O* 或图 4-2 中间的小格。

4.2.2 像素最佳预测模式的判定

在计算出 *dx* 和 *dy* 之后，就可以按公式(4.1)计算出边缘方向角，然后和各种预测模式的预测角度相比较，找出一个最接近的作为该像素的最佳预测模式。这一过程实际上相当于将方向角量化为某一个预测角度。

图 4-3 以 Intra_4x4 模式为例图解了最佳预测模式的判定过程，箭头代表了 8 种 Intra_4x4 预测方向，旁边的数字是模式号。对于这 8 种预测方向，可以在每两个相邻的方向之间划定一个边界，如图 4-3 中的虚线所示，称边界线所具有的角度为边界角度，具体实现时，边界角度取相邻预测角度的中值。每两个相邻的边界定义了一个边界角度范围，简称角度范围，一个角度范围与一个预测模式相对应。当一个方向角落在某个角度范围内时，与该角度范围相对应的预测模式就被认为是该像素的最佳预测模式。例如模式 3 的角度是 45°，模式 8 的角度是

$\arctan 0.5 \approx 26.5651°$，模式 7 的角度是 $\arctan 2 \approx 63.4349°$，则模式 3 与 8 的边界角度大约为 35.78°，模式 3 与 7 的边界角度大约为 54.2°，这样如果有个像素的方向角为 40°，它落在 35.78°与 54.2°之间，则认为这个像素的最佳预测模式为模式 3。

图 4-4 图解了 Intra_16x16 和 chroma 模式下最佳预测模式的判定。请注意，这里 plane 模式的角度范围与文献[40]定义的不一样。对于角度范围 $(-67.5°,-22.5°]$,由于没有对应的预测方向，所以只得指定为 DC 模式。

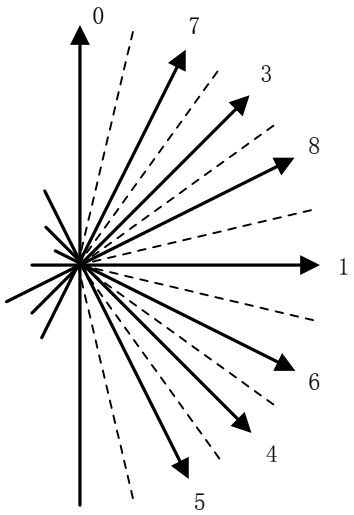


图 4-3 Intra_4x4 模式下像素最佳预测方向的判定

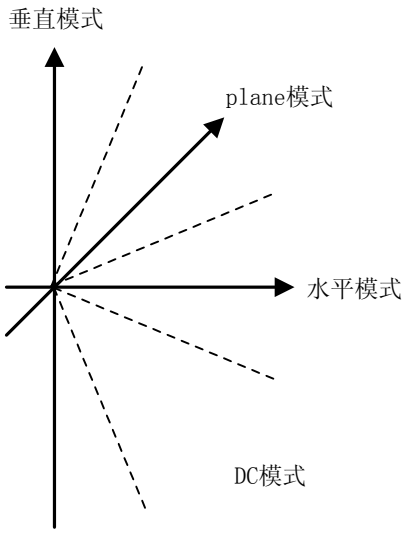


图 4-4 Intra_16x16 和 chroma 模式下最佳预测方向的判定

4.2.3 正切范围

如果直接根据公式(4.1)来计算方向角并与各个预测方向相比较，将增加设计上的复杂度，因为这需要求反正切函数。为了避免求反正切函数，这里将角度的比较转化为正切函数之间的比较。

根据公式(4.1)，方向角 θ 的正切应为

$$\tan\theta = \frac{dx}{dy} \tag{4.4}$$

由于 θ 的取值范围为 $(-90°, 90°]$，在此范围内 $\tan\theta$ 是单调递增的，所以如果有个方向角落在某个角度范围内，那么这个方向角的正切也应该落在对应的正切范围内。这里的正切范围指的是由边界角度的正切确定的一个范围。

图 4-5 示例了通过正切范围确定最佳预测模式的过程。各个预测模式的预测角度、边界角度范围、正切范围示于表 4-1。需要说明的是，对于垂直方向（模式 0），只比较正切的绝对值，因为图像的边缘方向朝上或是朝下是一回事，而 $\tan\theta$ 则会取不同的符号。

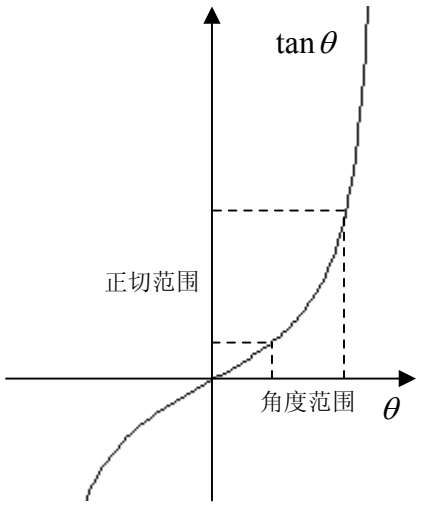


图 4-5 通过比较正切来确定最佳预测方向

表 4-1 各个预测模式的预测角度、边界角度范围和正切范围

模式		预测角度	边界角度范围	正切范围
Intra_16x16	0（垂直）	90°	[–90°,–67.5°] , (67.5°,90°]	绝对值大于 2.4142
	1（水平）	0°	(–22.5°,22.5°]	(-0.4142,0.4142]
	2（DC）		(–67.5°,–22.5°]	[-2.4142,-0.4142]
	3（plane）	45°	(22.5°,67.5°]	(0.4142,2.4142]
Intra_4x4	0（垂直）	90°	[–90°,–76.7175°] , (76.7175°,90°]	绝对值大于 4.2361
	1（水平）	0°	(–13.2825°,13.2825°]	(-0.2361,0.2361]
	3	45°	(35.7825°,54.2175°]	(0.7208,1.3874]
	4	-45°	(–54.2175°,–35.7825°]	(-1.3874,-0.7208]
	5	-63.4349°	(–76.7175°,–54.2175°]	[-4.2361,-1.3874]
	6	-26.5651°	(–35.7825°,–13.2825°]	(-0.7208,-0.2361]
	7	63.4349°	(54.2175°,76.7175°]	(1.3874,4.2361]
	8	26.5651°	(13.2825°,35.7825°]	(0.2361,0.7208]
chroma	0（DC）		(–67.5°,–22.5°]	[-2.4142,-0.4142]
	1（水平）	0°	(–22.5°,22.5°]	(-0.4142,0.4142]

	2（垂直）	90°	[–90°,–67.5°]，(67.5°,90°]	绝对值大于2.4142
	3（plane）	45°	(22.5°,67.5°]	(0.4142,2.4142]

4.2.4 直方图

为便于叙述，下面将像素的最佳预测模式简称为像素的模式。前面介绍了如何找出某一像素的模式，那么对一个预测块整体而言，应该如何判断它的预测模式呢？这里我采用直方图来判定，其原理就是统计当前预测块中哪种模式的像素数最多，就把哪种模式作为当前预测块的候选预测模式。

图 4-6 是一个 Intra_4x4 的直方图，图中横坐标是 8 个预测模式（注意没有 DC 模式，因为该模式不具有方向性），纵坐标是当前 4x4 大小的块中以该模式为最佳预测模式的像素数。由于在一帧的边缘处，Sobel 算子不可用，所以位于一帧边缘处的像素不参与统计。这里统计的是像素数，而非文献[40]所使用的方向矢量的幅值（ΣAmp），这是因为我考虑到如果当前块中有少数个别像素的模式并不占大多数，但由于噪声导致其方向矢量幅值非常大，那么取直方图最大值时就有可能选中这些像素的模式，而这些个别像素我认为是不能够代表整个预测块的图像边缘方向信息的。另外我的方法也省去了计算方向矢量幅值的工作，可以进一步加快程序的运行。在统计直方图的时候，算法还判断哪些模式在当前块中是可用的，对于不可用模式所对应的像素，不进行统计。例如位于一帧顶部的块，垂直方向的预测模式就用不了。

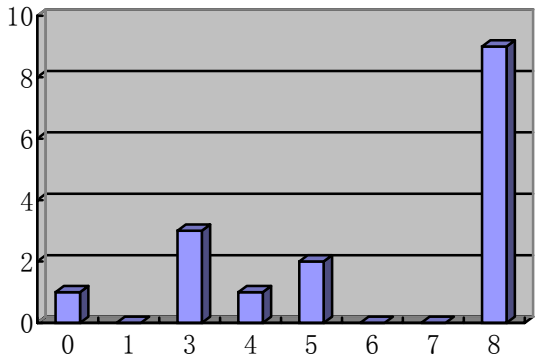


图 4-6 Intra_4x4 模式下直方图的一个例子

4.2.5 预测块候选模式的确定

这一步的目的是找出预测块的候选模式。这里的预测块可以是 Intra_4x4 模式下 4x4 大小的 luma 块，可以是 Intra_16x16 模式下 16x16 的 luma 块，也可以是 8x8 的 Cr 或 Cb 块。在具体实现时，为了有较大的选择余地，保证一定的图像质量，根据直方图找出的预测块候选模式可能不止一个。具体步骤如下。

步骤一：按 4.2.4 节统计当前预测块的直方图。

步骤二：判断候选模式是否采用 DC 模式。如果当前块比较均匀，没有明显的边缘特征，那么反映到直方图上，各个柱子的高度也应该相差不大，这可以用直方图的方差来判断：

$$DH = EH^2 - (EH)^2 \tag{4.5}$$

其中 H 表示直方图的数据，E 表示取均值。将计算出来的方差与预先设置的一个阈值进行比较，如果小于这个阈值，则采用 DC 模式，否则继续。三种预测块分别有三种不同的阈值。

步骤三：首先需说明一点的是，到了这一步，直方图中肯定有大于零的数据，否则方差为 0，在步骤二就结束了。对于 Intra_16x16 块和 Cr, Cb 块，将直方图中最大值所对应的模式作为该块的候选模式，Intra_16x16 块和 Cr, Cb 块的候选模式最多只有一种。

对于 Intra_4x4 块，找出直方图数据中不等于零的最大三个值，作为候选模式。与文献[40]不同的是，这里不是用最大值及相邻的两个模式作为候选，其原因还是因为我认为采用什么模式的决定因素是像素数。如果直方图的数据中不等于零的值不足三个，则只将大于零的值所对应的模式作为候选模式。Intra_4x4 块的候选模式最多为三种。

4.2.6 宏块最佳预测模式的确定

在前面确定了各种预测块的候选模式之后，就可以根据它们确定宏块的编码模式了。对于 Intra_4x4 块，由于最多可以有三种候选模式，所以首先按照 RDO 算法在三个候选模式中找出一个 R-D 意义上最优的模式，然后将这 16 个最优的 4x4 候选模式与 16x16 的候选模式合在一起作为 luma 分量的候选模式。对于

chroma 分量，算法同时考虑了 Cr 和 Cb 块的候选模式，将其作为 chroma 分量的候选模式。接着，对于一个宏块 MB，算法将 luma 分量的候选模式与 chroma 分量的候选模式相组合，根据 RDO 算法找出一个 R-D 意义上最优的模式组合，按照这个模式对 MB 进行实际编码。这样，一个 MB 需要进行 RDO 计算的模式组合数最多仅为(16*3+1)*2=98 种，远远小于原先的 592 种，所以该算法可以达到大幅减低运算量，提高运行速度的目的。

4.3 实验结果

这里我分别采用上面介绍的快速帧内预测模式选择算法与穷举 592 种预测模式的常规算法，对 foreman 序列的 400 帧图像全部按 I 帧进行编码。图 4-7 是快速算法与常规算法所用时间的对比，其中横轴表示不同的 QP，纵轴是编码 400 帧所耗时间，单位为秒。平均下来，快速算法的时间为常规算法的 29.92%。因为不同的机器所用的时间不同，这里就不详细列出具体的时间了。

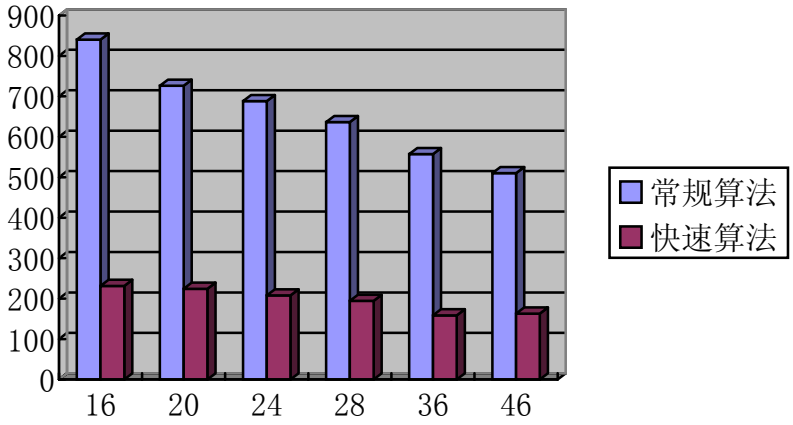


图 4-7 快速算法与常规算法时间对比

这个快速算法对帧内编码速度的提高是以编码性能的下降为代价的。图 4-8 是快速算法与常规算法的 R-D 曲线，由图中可以看出，虽然快速算法的性能较之常规算法有所下降，但下降的程度是很小的。因此这里可以得出结论：本文中所介绍的基于图像边缘方向检测的帧内预测模式选择的快速算法，能够在性能下降很小的情况下，显著地提高帧内编码的时间。

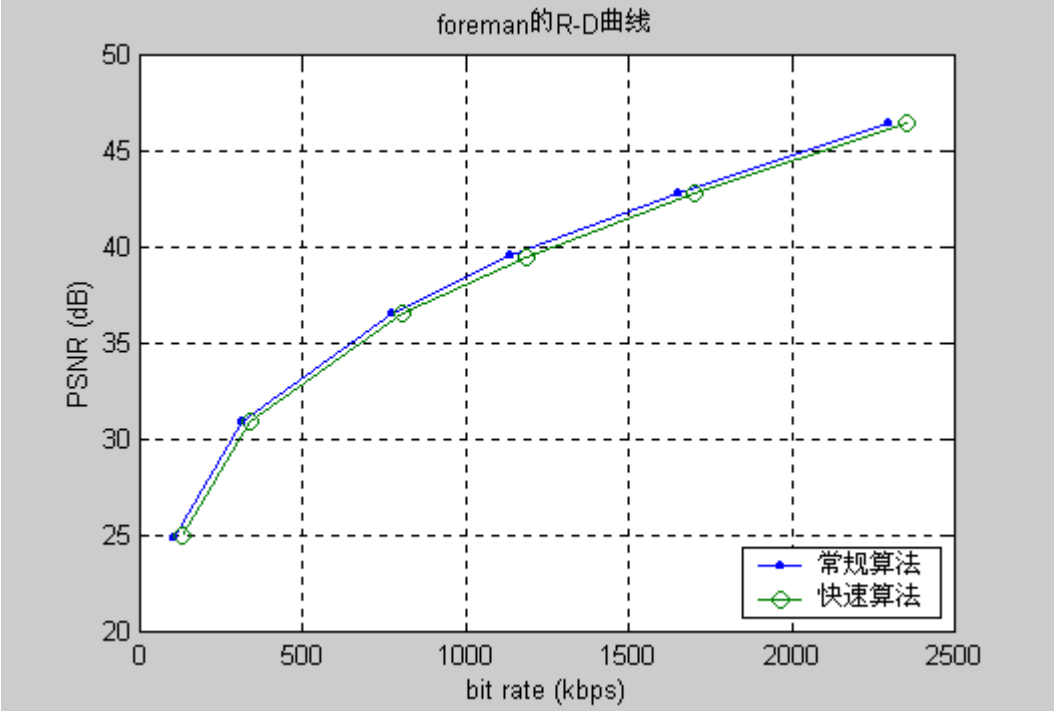


图 4-8 快速算法与常规算法的性能比较

致谢

在论文的最后，我要由衷地向一些曾经帮助过我，给予我支持的人表示感谢。

首先我要感谢我的导师李晓飞副教授，他在学习与研究中给了我悉心的指导，尤其是李老师严谨求实的治学态度、精益求精的工作精神更使我受益非浅。这里特别要提到的是在论文的选题及研究过程中李老师始终给了我极大的学术自由，让我可以尽情享受研究自己喜爱的课题的乐趣，所以这里我要首先感谢李老师对我的支持与帮助。

我要感谢龚建荣副教授，在南邮的三年中龚老师一直无微不至地关心着我们的学习与生活，在教研室的日常事务上也给予了我们极大的方便，这为我能够顺利地进入论文阶段提供了许多物质上的保证。另外，他平易近人的和蔼作风给我留下了深刻的印象，他对我论文的指导与帮助我也将铭记于心。

这里我特别要感谢贝璟同学，因为正是她向我介绍了 H.264 标准，引导我走上了一条充满无穷乐趣与绚丽美景的探索之路，虽然有时很艰辛，但一个又一个发现的喜悦无时不鼓励着我继续前进。我很希望能把这份喜悦与贝璟同学分享，也许这是对她给我的帮助最好的回报。

有很多人我要感谢，但我惟恐因为疏忽漏掉了某个人的名字，所以这里我就不一一列出他们的名字了。但我要对这些人说：感谢你们，没有你们的支持与帮助，就没有这篇论文，就没有我今天的收获，愿你们永远幸福平安！

参考文献

[1] "Editor's Proposed Draft Text Modifications for Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC), Draft 2", JVT Document: JVT-E022d2, October 2002

[2] "Study of Final Committee Draft of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)", JVT Document: JVT-F100, December 2002

[3] "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)", JVT Document: JVT-G050, March 2003

[4] JM Reference Software 6.1e, ITU-T, March 2003

[5] Nokia Research Center, "Proposal for H.263 requirements", VCEG Document: Q15-B-35, September 1997

[6] "Draft Requirements for H.263L - revised reflecting the Sunriver meeting results", VCEG Document: Q15-B-70, September 1997

[7] "Report of the Third Meeting (Meeting C) of the ITU-T Q.15/16", VCEG Document: Q15-C-48, December 1997

[8] Till Halbach, "H.26L Tutorial – FCD"

[9] "Liaison Statement to ITU-T SG 16 regarding the Joint Video Team", VCEG Document: VCEG-N12, July 2001

[10] UB Video Inc, "Emerging H.26L Standard: Overview and TMS320C64x Digital Media Platform Implementation", white paper, Feb 2002

[11] VideoLocus Inc, "VideoLocus VLP4000 Video Evaluation Platform for H.264/MPEG-4 AVC", datasheet, 2002

[12] Yun Q. Shi, Huifang Sun, "Image and video compression for multimedia engineering: fundamentals, algorithms, and standards", CRC Press, Boca Raton, 2000

[13] Thomas Wiegand, Xiaozheng Zhang, Bernd Girod, "Long-Term Memory Motion-Compensated Prediction", IEEE Trans. Circ&Syst. Video Tech., vol. 9, no. 1, Feb 1999, pp. 70-84

[14] 傅祖芸，“信息论基础”

[15] ISO/IEC JTC 1/SC29/WG1 N1646R "JPEG2000 Part I Final Committee Draft Version 1.0", March 2000

[16] Iain E G Richardson, "H.264/MPEG-4 Part 10 White Paper", 2002-2003

[17] Marta Karczewicz, Ragip Kurceren, "A Proposal for SP-frames", VCEG Document: VCEG-L27, January 2001

[18] Anthony Joch, Faouzi Kossentini and Panos Nasiopoulos, "A performance Analysis of the ITU-T Draft H.26L Video Coding Standard"

[19] Guy Côté, Lowell Winger, "Recent Advances in Video Compression Standards", IEEE Canadian Review – Spring/Printemps 2002

[20] Vivek K Goyal, "Theoretical Foundations of Transform Coding", IEEE Signal Processing Mag., pp. 9-21, Sep 2001

[21] 胡广书，“数字信号处理”，清华大学出版社，1997

[22] Antti Hallapuro, Marta Karczewicz, Henrique Malvar, "Low Complexity Transform and Quantization – Part I: Basic Implementation", JVT Document: JVT-B038, Jan 2002

[23] 吴成柯，戴善容，陆心如，“图象通信”，西安电子科技大学出版社，1990

[24] Minhua Zhou, "Modified Universal Variable Length Coding", VCEG Document: VCEG-N036, September 2001

[25] "JVT (of ISO/IEC MPEG and ITU-T Q.6/16 VCEG) 2nd Meeting Report, Jan 29 – Feb 1 2002", JVT Document: JVT-B001, Feb 2002

[26] Louis Kerofsky, Minhua Zhou, "Reduced Complexity VLC", JVT Document: JVT-B029, Jan 2002

[27] Gisle Bjøntegaard, Karl Lillevold, "Context-adaptive VLC (CVLC) coding of coefficients", JVT Document: JVT-C028, May 2002

[28] James Au, "Complexity Reduction for CAVLC", JVT Document: JVT-D034, July 2002

[29] "Editor's Proposed Draft Text Modifications for Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)", Draft 4, JVT Document: JVT-E022d4, October 2002

[30] Detlev Marpe, Gabi Blättermann, Thomas Wiegand, "Adaptive Codes for H.26L", VCEG Document: VCEG-L13, January 2001

[31] René J. van der Vleuten, "Low-Complexity Arithmetic Coding for CABAC", JVT Document: JVT-C029, May 2002

[32] Detlev Marpe, Guido Heising, Gabi Blättermann, and Thomas Wiegand, "Fast Arithmetic Coding for CABAC", JVT Document: JVT-C061, March, 2002

[33] Gary J. Sullivan, Thomas Wiegand, "Rate-Distortion Optimization for Video Compression", IEEE Signal Processing Mag., pp. 74-89, Nov 1998

[34] Gary Sullivan, "Recommended Simulation Common Conditions for H.26L Coding Efficiency Experiments on Low-Resolution Progressive-Scan Source Material", VCEG Document: VCEG-N81 draft1, Sep 2001

[35] 张志涌 等编著，“精通 MATLAB 5.3 版”，北京航空航天大学出版社，2000 年 8 月

[36] Jie Chen, Ut-Va Koc, and K. J. Ray Liu, "Design of Digital Video Coding Systems", Marcel Dekker, Inc., New York, 2002

[37] Xiang Li, Guowei Wu, "Fast Integer Pixel Motion Estimation", JVT Document: JVT-F011, December 2002.

[38] Zhibo Chen, Peng Zhou, Yun He, "Fast Integer Pel and Fractional Pel Motion Estimation for JVT", JVT Document: JVT-F017, December 2002

[39] Hye-Yeon Cheong Tourapis, Alexis Michael Tourapis, Pankaj Topiwala, "Fast Motion Estimation within the JVT codec", JVT Document: JVT-E023, October 2002

[40] Feng PAN, Xiao LIN, et al, "Fast Mode Decision for Intra Prediction", JVT Document: JVT-G013, March 2003

[41] Kenneth R. Castleman, "Digital Image Processing", 朱志刚 等译，Prentice Hall 出版公司，电子工业出版社，1998