北京交通大学

硕士学位论文

基于H.264的熵编解码的研究及优化

姓名：钮任飞

申请学位级别：硕士

专业：通信与信息系统

指导教师：荆涛

20090601

中文摘要

摘要：H.264 作为新一代的视频编码标准，代表了近期视频通信领域的研究水平。较之于以往的视频编码标准，H.264 在视频数据压缩效率和网络适应性等各方面都具有很好的性能。然而，随着性能的提升，H.264 的算法复杂度较以往视频标准也较高，这在一定程度上影响了 H.264 在一些实时性要求较强的场合的应用。基于上下文的自适应可变长编码(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)是 H.264 中采用的较为先进的熵编解码技术。与普通的熵编解码技术相比，基于上下文的自适应可变长编码和基于上下文的自适应二进制算术编码因在编解码过程中充分利用了上下文信息而进一步提高了数据的压缩效率。但是这两种熵编解码技术复杂的编解码过程增加了 H.264 的编解码时间，降低了 H.264 的编解码速度，对两种熵编解码算法进行优化来提高其编解码速度，提高 H.264 的实时性可以促进 H.264 在实际视频通信中的应用。

本文首先对 H.264 中的帧内帧间预测编码、变换与量化等关键环节进行了简要介绍，然后对 H.264 中的熵编解码——基于上下文的自适应可变长编码和基于上下文的自适应二进制算术编码进行了详细分析，并对其优缺点进行了研究、总结，最后针对部分缺点对基于上下文的自适应可变长编码的解码过程和编码 I 条带宏块类型时基于上下文的自适应二进制算术编码的编码过程进行了优化与改进。

针对基于上下文的自适应可变长编码的缺点，本文统计了解码过程中参数 NC 的取值情况，以及解码非零系数个数和拖尾系数个数时读入的比特数。统计数据表明，NC=6 的情况在解码过程中相对较少，同时解码非零系数个数和拖尾系数个数时读入的比特数少于等于 3 的情况居多。基于统计结果，本文对基于上下文的自适应可变长编码的解码过程进行了优化。实验数据表明，优化后的算法解码时间较原算法缩短了 1%左右。

针对基于上下文的自适应二进制算术编码的缺点，本文对编码 I 条带宏块类型时基于上下文的自适应二进制算术编码的编码过程进行了改进。实验数据表明，改进后的算法较原算法对相关编码函数的调用次数减少了 20%左右，同时编码时间缩短了 3%左右。

通过对两种熵编解码算法的改进，H.264 的编解码实时性得到了进一步提高，这对于一些实时性要求较高的场合是十分重要的。

关键词：H.264；基于上下文的自适应可变长编码；基于上下文的自适应二进制算术编码；JM11.0

分类号：TN919.81

ABSTRACT

ABSTRACT: H.264 is a new video coding standard on behalf of the recent research in the field of video communication. Compared to previous video coding standards, H.264 has better performance on video data compression and network adaptability. However, the algorithm complexity of H.264 encoding and decoding is higher, which to some extent, affects the application of H.264, especially in some real-time occasions. Context-based Adaptive Variable Length Coding(CAVLC) and Context-based Adaptive Binary Arithmetic Coding(CABAC) are advanced entropy coding methods used in H.264 to increase the degree of data compression due to their use of context information compared to common entropy coding methods. However, the complex processes of these two methods increase the encoding/decoding time of H.264 and reduce its speed. Optimization on the algorithm of these two entropy coding methods can speed up the H.264 encoding/decoding and promote its application in actual applications.

In this paper, some key aspects of H.264 including intra/inter frame prediction, transform and quantization are introduced at first, then Context-based Adaptive Variable Length Coding and Context-based Adaptive Binary Arithmetic Coding are analyzed and their advantages and disadvantages are studied. Finally the decoding process of Context-based Adaptive Variable Length Coding and encoding process of Context-based Adaptive Binary Arithmetic Coding while encoding macroblock type in I slices are optimized.

For Context-based Adaptive Variable Length Coding, the values of the parameter NC and the number of bits used to decode Coeff_token are collected and studied. The results show that the probability of NC = 6 is relatively low and the number of bits used to decode Coeff_token is more probable less or equal to 3.Based on the statistical results, an optimization for Context-based Adaptive Variable Length Coding is proposed in this paper. After optimization, the decoding time can be reduced by about 1% compared to the original algorithm.

For Context-based Adaptive Binary Arithmetic Coding, an improved algorithm to encode macroblock type in I slices is proposed. The results show that the calls of encoding function can be reduced by about 20% in the improved algorithm and the encoding time can be reduced by about 3%.

After improvement on the two entropy coding methods, the real-time performance

of H.264 is better than before, which is very important in some real-time occasions.

KEYWORDS: H.264; Context-based Adaptive Variable Length Coding; Context-based Adaptive Binary Arithmetic Coding; JM11.0

CLASSNO: TN919.81

学位论文版权使用授权书

本学位论文作者完全了解北京交通大学有关保留、使用学位论文的规定。特授权北京交通大学可以将学位论文的全部或部分内容编入有关数据库进行检索，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。同意学校向国家有关部门或机构送交论文的复印件和磁盘。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：钮佳飞　　导师签名：[illegible]涛

签字日期：2009年6月16日　　签字日期：2009年6月16日

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作和取得的研究成果，除了文中特别加以标注和致谢之处外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京交通大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者签名：钮伟飞 签字日期：2009 年 6 月 16 日

致谢

本论文的工作是在我的导师荆涛教授的悉心指导下完成的，荆涛教授广博的学识、敏锐的洞察力、严谨的治学态度和科学的工作方法给了我极大的帮助和影响。荆老师求真务实、一丝不苟的工作作风使我敬佩!在论文期间，荆老师的不断鼓励和在生活上的关怀使我的精神倍受鼓舞。在此衷心感谢两年来荆涛老师对我的关心和指导。

时光飞逝！两年的硕士生活即将结束，衷心感谢王智、李兴华、卢燕飞、翟美云老师在此期间对我的关心和指导！

特别感谢实验室的霍炎博士，霍炎师兄无私的指导和耐心的帮助使我克服了学习和工作中的困难，顺利完成了论文。

感谢实验室的方君丽、魏媛珍、张玉等同学，与你们真挚的友谊和愉快相处使我每天都能保持一个好心情。

感谢各位评委老师在百忙中抽出时间来评审论文和参加答辩！

衷心感谢我的父母，你们在物质上的帮助和精神上的鼓励使我在生活和学习的道路上不断地前进，顺利完成了学业！

最后，感谢每一位曾经帮助过我的人！

1 引言

1.1 本课题的研究背景及意义

熵编码作为一种优良的数据压缩方法，在视频数据的各种压缩技术中得到了广泛的应用。1948 年香农信息论的创立，为以后熵编码技术的发展提供了理论基础。在香农信息论中，香农提出了信源熵的概念，并指出信源熵即为信源数据压缩的理论极限，可以通过不同的压缩方法去接近信源的熵。基于此，人们提出了若干不同的熵编码方法。1952 年，霍夫曼为了解决一阶马尔科夫序列的无失真压缩，提出了一种基于二叉树的数据压缩方法，人们称其为霍夫曼编码。霍夫曼编码通过精心设计相异字头的码字，根据信源符号出现情况的不同设定了不同码字的长度，从而实现了对数据的压缩。然而，霍夫曼编码的最小编码单位为比特，如果某个信源符号根据香农信息论计算其平均信息量为 0.2 比特，则对其进行霍夫曼编码则至少需要 1 比特，这就造成了数据压缩效率的降低。针对这种情况，1963 年 Peter Elias 提出了比霍夫曼编码更优的基于比特串的算术编码。但是直到 1976 年，Rissanen 和 Pasco 才给出了第一个可用的算术编码算法。与霍夫曼编码相比，算术编码更能接近信源的熵，因而其压缩效率较霍夫曼编码要高，不过相应地，其算法也较霍夫曼编码复杂。霍夫曼编码和算术编码都是基于概率统计特性的熵编码方法，可以逼近熵下界。

一般来说，如果待编码的数据之间具有很强的相关性，则对其直接进行熵编码压缩效率很低。因此考虑到视频数据相关性较强的特点，大部分视频编码标准都对视频序列的符号进行了统计，然后依据视频序列的统计特性制定合适的熵编码方法。早期的视频编码标准中，视频序列的统计特性是根据典型的视频序列进行统计得到的。然后根据此统计特性去设计 VLC 码表。通常情况下,码表中码字的长度都是固定的，这就造成了熵编码压缩性能的下降。因为显然不同视频序列中信源符号的概率是不同的。在最新的视频编码标准 H.264 中为避免这种情况，采用了基于上下文的自适应可变长编码和基于上下文的自适应二进制算术编码[1]。它们充分考虑了视频中的上下文信息，利用已编码的符号来为当前编码符号选择合适的上下文模型，实现了概率的实时更新，提高了压缩效率。但是这两种熵编解码算法复杂度高，耗时大。这对于 H.264 的实时应用极为不利。熵编码作为视频编码过程的最后一步，其好坏直接影响着视频压缩的效率和性能。为了提高 H.264 的实时性，对 H.264 中基于上下文的自适应可变长编码和基于上下文的自适应二

进制算术编码进行研究优化有着重要的意义。

1.2 视频数据压缩

1.2.1 视频数据冗余度分析

视频出于比语言、文字和声音更直观而成为人们获取信息的主要途径。自从1948年提出视频数字化至今，视频编码压缩技术已经经历了60多年的发展。随着多媒体、网络和通信技术的发展，人们对视频的关注也越来越高，因而视频压缩技术已经成为人们研究的热点[2]。目前，国际标准化组织(ISO)以及国际电信联盟(ITU-T)已经制定了多个视频编码标准。最新的H.264标准是由这两个组织共同制定而成。如今视频技术已经在视频存储、广播电视、因特网或无线网上的流媒体、电视会议等多个领域得到了广泛应用[3]。

人们通过对视频图像进行研究发现，原始视频数据中存在着大量冗余信息。去除这些冗余信息对视频质量并不会产生太大影响。这就使视频压缩和传输成为了可能。视频信号的冗余种类很多[4],一般来说包括：

空间冗余

图像中像素的灰度值是连续变化的，除边缘情况外像素和像素之间在行方向和列方向都具有较强的相关性。一个像素的灰度值可以由其相邻像素的灰度值预测出来。在实际视频压缩中可以通过相邻像素的预测来去掉这些冗余信息而实现数据的压缩。

时间冗余

视频中物体的运动一般是连续的，而且帧与帧之间的时间间隔一般来说很小，相邻帧或相近帧之间的像素有相当一部分是相同的，只不过是位置发生了变化而已。如果每帧都分别编码势必会对一些信息进行重复编码，这就造成了视频图像中时间上的冗余。在实际视频压缩中可以通过帧间预测来去掉这些冗余信息而实现数据的压缩。

知识冗余

有些图形图像与人们平时的知识积累有一定的相关性，只需要给出一些参数就可以勾画出来，而并不需要全部对图像去进行编码。此时可以只对这些特征参数去编码从而实现数据压缩。

视觉冗余

生理实验数据表明，人类眼睛对所有视频分量的敏感度并不相同，对低频失真比对高频失真更为敏感。而且对复杂区域的失真也不敏感。而原始视频数据中

对所有视频数据都是同等对待的，这就造成了基于人眼视觉系统的数据冗余，有效地去除这些冗余可以实现视频数据的压缩。

熵编码冗余

根据香农信息论的基本原理，设一个信源的输出是一个离散型变量{ a_1，a_2，…，a_n }，每个符号出现的为 $P(a_i)(i=1,2\ldots\ldots,n)$，且

$$\sum_{i=1}^{n} P(a_i)=1 \tag{1}$$

则每个符号的平均信息量为

$$I(a_i)=-\log_2(a_i) \tag{2}$$

整个信源所含的信息量可表示为

$$H=-\sum_{i=1}^{n} P(a_i)\log_2(a_i) \tag{3}$$

称为熵。信源的熵值是信源所含信息量的一个理论极限，如果编码码长大于信源的熵，则表明此编码方式存在冗余。编码码长越接近于信源的熵，则此种编码方式的冗余度越低，此种编码方式的压缩效率也就越高。实际的编码过程都是通过选择不同的熵编码方式来去除这些冗余而达到数据压缩的目的。

1.2.2 视频数据压缩的基本概念

视频压缩就是通过各种数学计算方法以及计算机技术将视频图像数据中的冗余信息尽可能多地去掉，同时在解码压缩数据后能够尽可能地恢复原图像的内容。按照图像数据的压缩情况来分类，视频图像的压缩可以分为无损压缩和有损压缩两种。无损压缩就是在压缩过程中不改变图像数据的内容，在解码后能够完全恢复原图像数据的方法；而有损压缩就是在图像数据的压缩过程中为了获得更好的压缩比而去掉一些图像数据中的次要信息，在解码后不能完全恢复图像数据的压缩方法。这两种压缩方法可以适应于不同的应用场合：对于航天探测、医疗诊断等场合，可以利用无损压缩来完全的恢复视频图像数据；而对于一些娱乐观赏性视频图像数据则可以采用有损压缩来提高视频图像数据压缩效率，减小视频图像文件的体积，从而更利于数据的储存与传输。

1.2.3 视频数据压缩的一般过程

图 1 为视频图像数据压缩编码的一般过程框图，输入的当前帧数据通过与参考帧数据进行做差后残余数据被送入变换器进行变换，然后变换系数被量化，量化后的变换系数经过熵编码后输出。同时量化系数还被进行反量化、反变换来重构帧作为后续图像的参考帧。

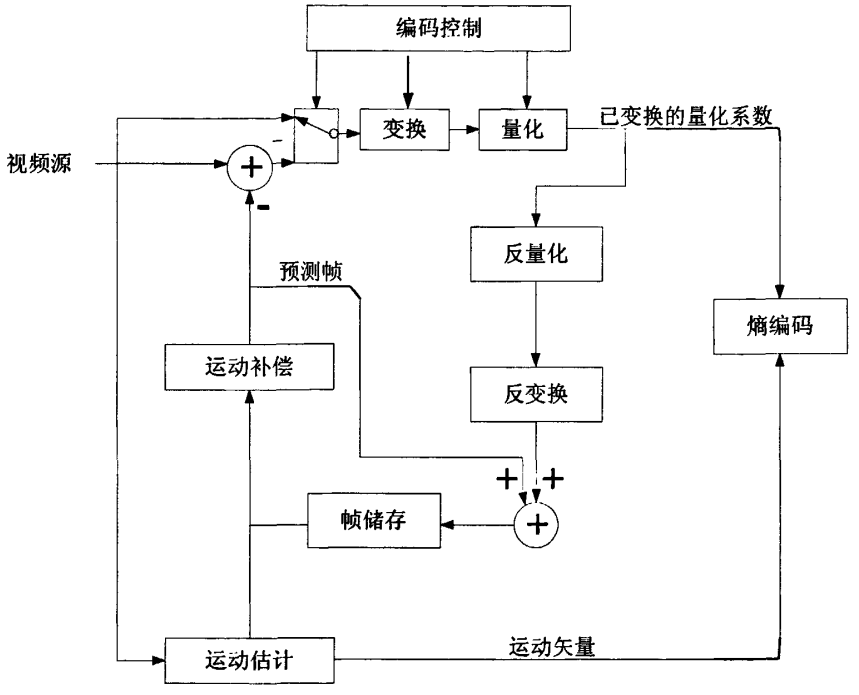

图 1 视频数据压缩编码的一般过程框图

Figure 1 General Video data compression process

1.2.4 图像视频编码标准介绍

静止图像压缩标准 JPEG 系列

JPEG[5]最早由 Joint Photographic Experts Group(中文译为联合图像专家组)于 1991 年 3 月提出草案。其基本原理是对图像数据中存在的冗余信息进行压缩，且是一种有损压缩格式。由于其品质优良，在其发布后的几年内得到了广泛的应用。在网络上或者数码摄像领域的很多地方都采用了该标准。在科研领域，JPEG 标准的提出也为以后的视频压缩标准奠定了基础。

JPEG 作为一种高效的图像压缩技术，其一个重要的特点就是压缩比很高[6]。，且根据不同的应用要求可以调节其压缩比例。其压缩比例的可调范围很宽，在 10：1 到 40：1 之间。当然，其压缩后的图像质量也是随着压缩比例的增大而下降的，当压缩比例太高时，重建后的图像就会呈现比较明显的块效应。同时，JPEG 也不支持无损图像压缩。因此，在 JPEG 的基础上，Joint Photographic Experts Group 又提出了 JPEG2000 标准。JPEG2000 作为 JPEG 的改进标准，其数据压缩率相比 JPEG 提高了 30%左右，且同时支持有损数据压缩和无损数据压缩，这使得 JPEG 系列的灵活性得到了进一步提高，因而其应用范围也得到了较大的拓展。该标准是由联合摄影专家组于 1997 年开始征集提案[7][8]，其第一部分在 2000 年 12 月正式公布，其他部分则在之后陆续公布。相比 JPEG 标准，JPEG2000 的优点包括：支持更高的压缩率、同时支持无损压缩和有损压缩、支持数据的渐进传输、支持感兴趣区域压缩、支持码流的随机访问和处理、提供更好的容错性等。

总之，JPEG 系列的出现使得视频图像压缩和网络视频数据传输得到了很大的改进，且为以后的视频编码标准提供了较好的参考模型。

运动图像压缩标准 MPEG 系列

MPEG 的全称为 Moving Picture Experts Group，即运动图像专家组，因此其研究领域主要运动视频图像的压缩。该标准组于 1988 年由国际标准化组织(ISO)成立，其制定的标准称为 MPEG 系列。目前 MPEG 系列标准主要包括：MPEG-1、MPEG-2、MPEG-4 等。下边将分别说明。

1、MPEG-1

MPEG-1[9]是 MPEG 最早制定的一个视频编码标准，该标准于 1992 年正式出版。其基本的数据传输数率为 1.5Mbits/sec，每秒播放 30 帧，它是 MPEG 的小画面模式，平均压缩比为 50：1。该标准在问世后不久就得到了商业应用，我们所熟知的 VCD 就采用了 MPEG-1 的编码格式。MPEG-1 同时也被应用于视频点播(VOD)、非对称数字用户线路(ADSL)等数字电话网络。其视频图像质量与 VHS 相当。

2、MPEG-2

MPEG-2 标准是继 MPEG-1 标准后的又一个视频编码标准，于 1994 年公布。在 MPEG-2 标准中第一次引进了档次(profile)与级别(level)的概念[10]，使得 MPEG 系列的互用性进一步增强，且其灵活性也得到了改善。MPEG-2 按压缩比大小的不同分成五个档次，每一个档次又按图像清晰度的不同分成四种级别。这样，不同的档次和级别可以组成 20 种不同的组合，常用的组合大约有 11 种左右。较之于 MPEG-1，MPEG-2 的视频图像清晰度得到了很大的提高，且兼容了 MPEG-1 标准。这使得 MPEG-2 在高清晰度视频压缩方面得到了应用，我们所熟知的 DVD

就利用了 MPEG-2 的视频编码方式，具有演播室质量标准清晰度电视 SDTV 中也利用了 MPEG-2 编码方式。

3、MPEG-4

MPEG-4 标准是由 MPEG 工作组于 1999 年出台，并在当年的 12 月份提供了第二版的最终草案。与前两个视频编码标准相比，MPEG-4 是一个基于对象的视频编码标准[11]。因而当视频画面中存在快速运动的对象或者视频的场景有很大的变化时，MPEG-4 可以达到很好的性能，这是前两个视频标准所不能及的。同时，由于 MPEG-4 采用了较为先进的编码方法，使得视频数据的压缩效率进一步得到了提高。其应用目标主要是针对低比特率下的多媒体通信、广泛的兼容性等问题，因而其图像质量不及MPEG-2。但MPEG-4更适用于交互AV服务以及远程监控[12]，同时在娱乐以及网络视频欣赏方面较前两个视频标准应用范围更广。

运动图像压缩标准 H.26X 系列

H.26X 系列是由国际电信联盟(ITU-T)制定的一套视频编码标准系列。国际电信联盟创建于 1993 年，其是由国际电报电话咨询委员会(CCITT)发展而来。H.26X 系列主要包括 H.261、H.263、H.264 等。下边将分别说明。

1、H.261

H.261 最初是为在 ISDN 上实现视频会议而设计的，是第一个可应用的视频编码标准。其又称为 p×64kbps 视听业务的视频编解码器，其中 p 可取 1 至 30 之间的数据。H.261 采用基于块的混合编码方案[13]，采用了 4:2:0 采样格式，可对 CIF 和 QCIF 分辨率的视频进行编码。对于 H.261 格式，编码具有剧烈运动内容的图像要比静止内容的图像质量差。

2、H.263

H.263 在技术上对 H.261 进行了较大的扩充，这使得 H.263 的应用范围较 H.261 更为广泛。相比 H.261，H.263 可支持更多的视频分辨率，如 SQCIF、4CIF 和 16CIF 等；其运动补偿技术也由 H.261 的全象素精度变为半像素精度，同时采用了 PB 帧编码。目前 H.263 已基本取代 H.261。

3、H.264

H.264 将在第二章进行详细介绍。

1.3 本论文的主要工作

本论文的主要工作是对视频压缩标准 H.264 编解码过程中的基于上下文的自适应可变长编码和基于上下文的自适应二进制算术编码两种熵编解码技术进行研究与优化。

针对基于上下文的自适应可变长编码，本文对其解码过程中耗时较大的环节进行了研究，利用 VC6.0 通过 JM11.0 模型统计了解码过程中部分参数的值，并根据统计结果对解码过程进行了优化。

针对基于上下文的自适应二进制算术编码，本文仔细分析了其编码过程，并对编码过程中影响编码实时性的一些环节进行了剖析，总结了其缺点，最后针对这些缺点对编码 I 条带宏块类型时的编码过程进行了改进。

通过实验对比，优化与改进后的算法编解码复杂度降低，实时性得到了提高。

1.4 论文的组织结构

全文共分为五章。

引言部分简要介绍了熵编解码技术的发展及在视频数据压缩中的应用、视频数据的冗余度以及几种不同的视频编码标准。随后介绍了论文的主要工作以及组织结构。

第二章对 H.264 编码过程中的主要环节进行了简要的介绍。

第三章对基于上下文的自适应可变长编码做了较为详细的阐述，对解码过程中的问题进行了一定的剖析，利用 JM11.0 模型对解码过程中的一些参数进行了统计并根据统计结果进行了分析，最后对解码过程进行了优化。

第四章对基于上下文的自适应二进制算术编码做了较为详细的阐述，对编码过程中的问题进行了一定的剖析，总结了其在编码过程中存在的一些缺点，然后对编码 I 条带宏块类型时基于上下文的自适应二进制算术编码的编码过程进行了改进。

第五章是对全文的总结与展望。

2 H.264编解码技术介绍

2.1 概述

H.264于2003年3月正式发布，是由ITU-T的VCGE和ISO/IEC的MPEG的联合视频组(JVT)共同开发的一个新的视频标准。所以H.264同时又是MPEG-4的第十部分。其基本编码原理同以往的视频编码标准类似。但在各个环节的细节上进行了一系列改进与创新，如统一的VLC符号编码、基于4×4块的整数变换、分层的编码语法以及高精度、多模式的位移估计等。这些措施使得H.264较其他标准具有更高的压缩效率[14]，例如在相同的重建图像质量下H.264比H.263节约50%左右的比特率。目前H.264支持3种档次[15]：基本档次(baseline profile)、扩展档次(extend profile)、主要档次(main profile)。每种档次可以支持一类比较广泛的视频应用。基本档次对视频会议等实时性要求较强的视频通信过程特别适合；扩展档次非常适合于在网络上传输视频流；而主要档次则更适合于视频存储等应用。H.264在算法上分为两层：视频编码层(Video Coding Layer,VCL)和网络提取层(Network Abstraction Layer,NAL)。前者主要负责视频内容的处理，而后者主要负责网络的分段格式封装数据，包括组帧、逻辑信道的指令等。H.264和H.261、H.263类似，也是基于块的混合编码方法[16]，将每一帧视频图像分成若干个16×16的像素块，称为宏块。大部分的视频编码操作都是针对宏块进行的。在空间冗余度压缩方面，H.264主要采用了帧内预测的方式，通过同一帧内相邻宏块对当前宏块的像素值进行预测形成预测宏块，并对当前宏块与预测宏块做差，然后对残差数据进行变换。在时间冗余度压缩方面，H.264主要采用了帧间预测的方式，通过在相邻帧或相近帧之间搜索类似宏块来对当前帧中的宏块进行预测并形成预测宏块，然后对当前宏块与预测宏块做差，之后对残差数据进行变换。

2.2 H.264系统介绍

H.264的算法可以分为视频编码层和网络提取层[17]。前者负责对视频数据进行高效的压缩，后者负责对视频流进行打包以适应网络的传输要求。如图2所示。

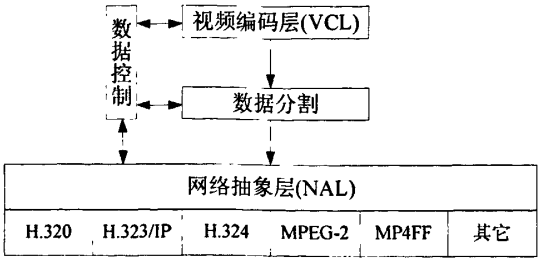

图2　H.264结构

Figure 2 Construction of H.264

视频编码层

在视频编码方面，H.264也是采用了基于宏块的混合视频编码方法。通过对宏块进行帧内帧间预测、变换、量化、重排序以及熵编码等环节来实现数据压缩过程。同时在一些细节方面又对一些方面做了改进来达到更好的压缩效率[18]。

网络提取层

NAL层由NAL单元所构成[19]。每个NAL单元第一个字节的头信息说明了NAL单元的数据类型。在H.264标准中分别定义了面向数据包和面向比特流的两种NAL单元结构的流格式。NAL单元可分为VCL单元和非VCL单元. 前者包含着视频数据中的抽样值，而后者包含着一些附加的相关信息。

2.3 H.264编解码原理

H.264标准中并没有定义视频编解码器，而是定义了生成视频流的详细编码语法以及解码方法。一般来说编码器应具有如图3所示的结构[20]。

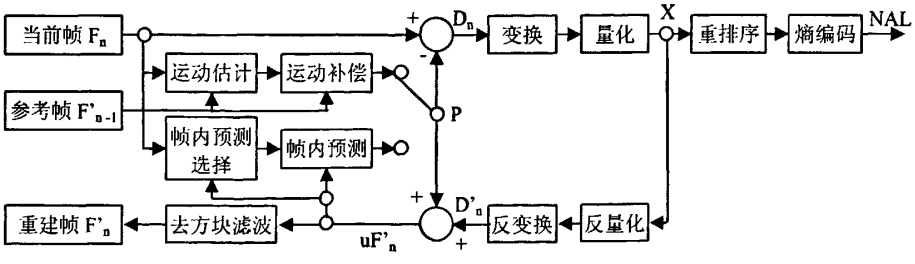

图3　H.264编码器框图

Figure 3 Block diagram of H.264

输入的当前帧经过与参考帧做差后形成残差数据，然后残差数据经过变换、量化、重排序以及熵编码后进行NAL封装。在残差数据进行量化后，同时在反馈回路又对其进行反量化、反变换、去方块滤波等环节进行帧重构，重构后的帧进

行储存以便于对后续帧进行帧间预测。

2.3.1　帧内预测

H.264中的帧内预测是在变换编码之前进行的。其运用了多种不同的编码方式来实现对数据的高度压缩，包括对4×4亮度块的9种预测模式以及16×16亮度块的4种预测模式。

而色度信号则采用8×8的4种预测方式[21]。这些预测模式充分利用了相邻像素间的数据相关性[22]。

预测模式说明

4×4亮度块的9种预测模式如图3所示。除边缘块外，当前块中的每个像素都可由块左上角的17个像素通过加权预测得到。这9种模式的详细描述见表1。

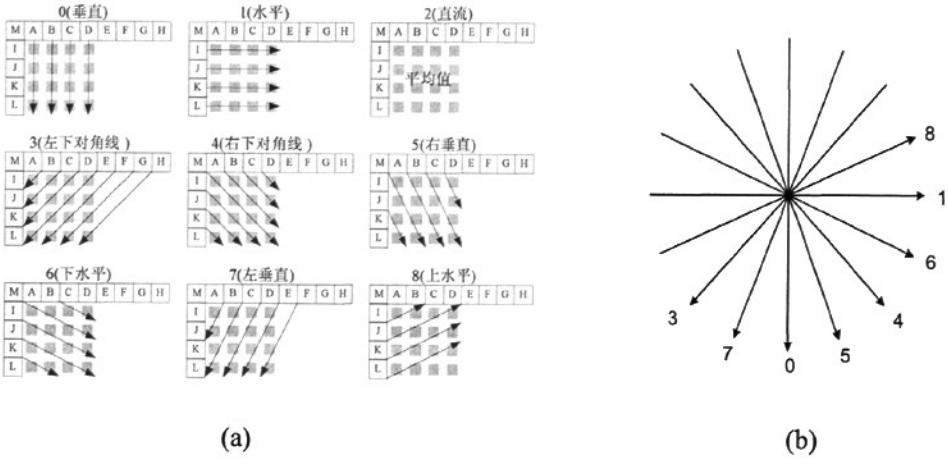

图3　4×4亮度块的9种预测模式(a)及方向图(b)

Figure 3 Nine kinds of prediction modes(a) and there directions(b) for 4×4 luma blocks

表1　4×4亮度块的9种预测模式

Table 1 Nine kinds of prediction modes for 4×4 luma blocks

模式	名称	备注
0	垂直预测	
1	水平预测	
2	直流预测	
3	下－左对角线预测	与模式0成45°角
4	下－右对角线预测	与模式1成45°角
5	垂直－右斜线预测	与模式2成22.5°角

表1　4×4亮度块的9种预测模式(续)

Table 1 Nine kinds of prediction modes for 4×4 luma blocks(continued)

模式	名称	备注
6	水平一下斜线预测	与模式1成22.5°角
7	垂直一左斜线预测	与模式0成22.5°角
8	水平一上斜线预测	与模式1成22.5°角

例如当利用模式3进行预测时，

$$\Pr ed(x,y) = Clip1(a + b(x-7) + c(y-7) + 16) >> 5 \tag{4}$$

其中

$$a = 16 \times (P(-1,15))、\ b = (5 \times H + 32) >> 6、\ c = (5 \times V + 32) >> 6 \tag{5}$$

函数$Clip1(x)$的定义为：若x<0,则$Clip1(x)$=0，若x>255,则$Clip1(x)$=255，否则，$Clip1(x)$=x.

$$H = \sum_{x=0}^{7} (x+1) \times [P(8+x,-1) - P(6-x,-1)] \tag{6}$$

$$V = \sum_{y=0}^{7} (y+1) \times [P(-1,8+y) - P(-1,6-y)] \tag{7}$$

其它的与此类似。

16×16亮度块的4种预测模式如图4所示。这种模式适用于亮度变化平缓的区域。

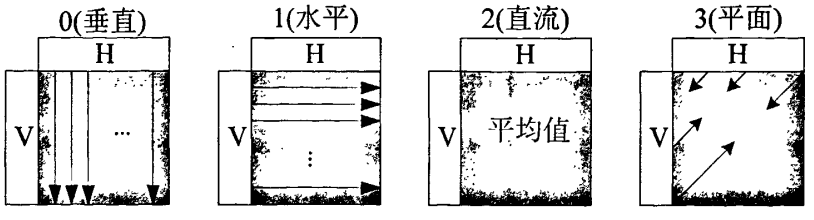

图4　16×16亮度块的4种预测模式

Figure 4 Four kinds of prediction modes for 16×16 luma blocks

H.264一般采用4:2:0的图像采样格式，这样每个16×16的亮度块就会对应于两个8×8的色度块。两个8×8的色度块采用与16×16亮度块类似的4种预测模式且两个色度块的预测模式相同。

预测模式编码

以4×4亮度块为例，每个块的预测模式如果都进行编码的话，势必会需要很多比特。H.264中充分利用了相邻块的相关性来编码预测模式。如图5所示，当对块C进行预测模式选择时，编码器会充分考虑相邻块A和B的预测模式。当A、B和C在同一条带时，C的预测模式即为A和B预测模式中的最小值，否则(A和B不同时可用)，C的预测模式将被设定为2(直流预测)。

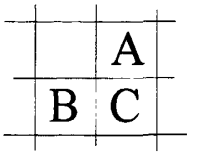

图5　相邻4×4块的帧内预测

Figure 5 Intra-frame prediction for neighbor 4×4 blocks

2.3.2　帧间预测

帧间预测用于降低视频帧之间的时域相关性。H.264中的帧间预测也是基于块结构的运动补偿。然而，它与早期的视频标准的区别在于H.264中将块进一步分割成多种结构的子块来进行预测，同时其运算精度进一步提高到四分之一像素。

树形结构的运动补偿及四分之一像素精度插值

在H.264中，每个宏块中的子块都会产生一个运动矢量并被编码传输，同时宏块的分割方式也会被编码传输。这样，如果选择大的分块进行预测，块的运动矢量以及分割方式需要编码的比特数较少，但是预测后的残差值较大，需要的比特数较多。反之，如果采用较小的块预测，残差值将会减小，编码比特数也随之减少，但是编码块的分割方式以及每个块的运动矢量所需的比特数就会增多。因此分块尺寸的选择对于数据的压缩影响很大。对于图像帧中变化比较平缓、细节内容比较少的区域，H.264中采用大的分块进行预测，而反之对于变化比较剧烈、细节内容较多的区域则选择小的分块进行预测。一个16×16的宏块可以有16×16、16×8、8×16、8×8等几种分割方式，而如果选择了8×8的分割方式，则还可以进一步划分为8×8、8×4、4×8、4×4等方式，如图6所示。

同样，每个亮度块中的色度分量(Cb分量和Cr分量)按照同样的分块方式进行

划分。把宏块分成不同尺寸的运动补偿子块的方法就是为了实现树形结构的运动补偿[23]。

虽然对块的大小进行了进一步的分割，但是有时候预测残差还不是很理想。对此H.264采用了另外一种方法来进一步减小预测残差来提高数据压缩效率，这就是四分之一像素插值。

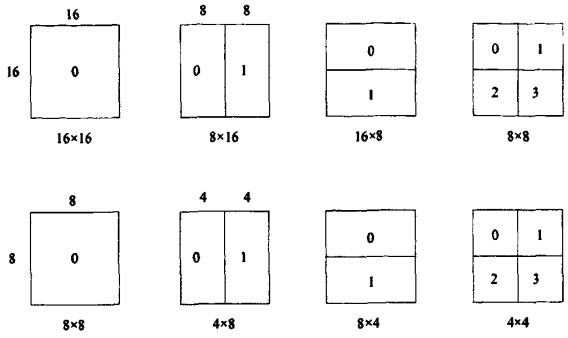

图6　H.264中宏块的划分

Figure 6 The division of macroblocks in H.264

H.264中通过使用一个六阶有限冲激响应滤波器来对相邻整数位置的像素值进行内插来得到内插的像素值，这些像素值可以更好地预测当前帧中的相关宏块，进一步减小预测残差。如图7所示，首先通过整数像素点来计算出半整数像素点的像素值，然后再利用半整数像素点和已知的整数像素点来计算出四分之一像素点的像素值。例如点b可以由下式得到(round表示取整)：

$$b=\text{round}((K-5L+20M+20N-5P+Q)/32) \tag{8}$$

当计算出所有半像素位置点的像素值后，四分之一点处的像素值便可由类似的方法得到。同理，对于采样方式为4:2:0的视频图像其色度分量为八分之一精度，其值由周围像素值内插得到。

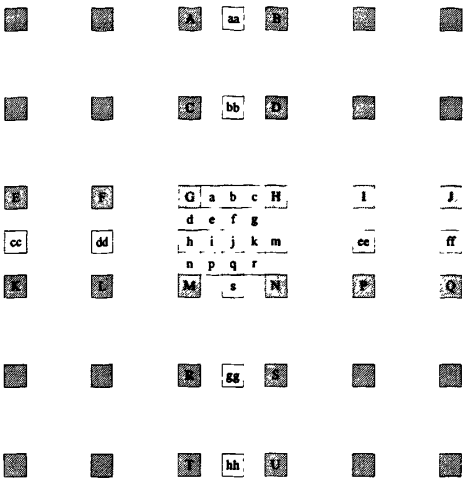

图 7　H.264 中像素点的四分之一插值

Figure 7 Quarter pixel interpolations in H.264

运动矢量预测

在 H.264 中每个块的运动矢量并不是被独立传输的，因为相邻块的运动矢量之间具有很强的相关性而且运动矢量的编码需要耗费大量比特。所以当前块的运动矢量是基于相邻块的运动矢量来进行预测得到残差并进行编码传输的。

2.3.3　变换和量化

为了对图像数据进行进一步的压缩，图像处理中常用变换编码和量化的方法去除图像中的相关性以及减小信号的动态范围。常用的变换为二维 DCT 变换。但在 H.264 中并不是直接使用二维 DCT 变换，而是使用了类似的整数变换，这是因为二维 DCT 变换具有以下不足之处：

1. 二维 DCT 变换中存在许多的无理数，这些无理数在实际运算过程中会被舍入成有理数，这不仅降低了运算速度，而且还增大了变换过程中的误差，使得反变换后的数据不能完全复原。
2. 二维 DCT 变换中存在大量的乘法运算，计算机的处理器在处理浮点数据的乘法运算时速度比较慢，这对于对实时性要求较高的视频传输来说是极为不利的。

基于以上不足之处，H.264 中提出了整数变换并将其与量化过程合并在一起，其基本原理类似于二维 DCT 变换，但是通过采取一些措施有效地避免了上述缺点。与二维 DCT 变换相比，整数变换所有的运算可与整数算术一起进行，没有降低精度，而且由于没有对数据进行舍入操作，所以不会存在编码器和解码器不匹配的

问题[24]。同时变换的核心部分有效地避开了乘法运算，只需要加减和移位就可以实现，这对于计算机来说实现起来较为快速。经过测试，整数变换的性能不逊色于DCT变换[25]。

在量化环节，H.264量化步长Q_{step}规定了52个可用值，如表2所示。QP是量化步长的序号，称为量化参数。QP取值越小，则量化越精细。应用时可以灵活选择量化参数。色度编码一般与亮度编码的量化步长相同。

表2　H.264中编解码器的量化步长

Table 2 Quantization steps in H.264

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

2.3.4　熵编码

熵编解码部分将在第三章中详细介绍。

2.4 本章小结

本章从整体上对H.264视频标准做了简单的介绍，对其中的一些关键技术进行了阐述，包括帧内帧间预测、变换和量化等环节，这些环节相对于以往视频编码方法的不同体现了H.264较其它视频编码的优越性。

3 CAVLC 编解码研究及解码优化

熵编码将描述视频流的符号串编码成适于传输和储存的比特流并对符号数据进行了压缩。它是 H.264 编码过程中的最后一个环节，也是编码环节中比较重要的一个环节。 H.264 针对不同的情况开发了不同的熵编码方法，其中基于上下文的自适应可变长编码和基于上下文的自适应二进制算术编码是两种常用的熵编码方式。在本章及接下来的第四章将详细探讨这两种编码方法并对其进行一些优化研究。

3.1 熵编码

熵编码的出现是基于香农的信息论[26]。在香农信息论中熵被作为一个重要概念来描述信号所含的信息量。参考公式(1)、(2)、(3)，若各个符号出现概率不同的话，则其携带的信息量也就不同，且符号出现的概率越小，其携带信息量就越大。为此，用公式(2)来表示单个符号的信息量[27]，则整个信源符号的信息量即为公式(3)，又称为信源的信息熵。信源的熵给出了信源数据压缩的极限，即信源熵是信源数据压缩的下限，不论通过何种压缩方法，压缩后的数据量都不可能小于信源的熵。基于香农的信息论，人们开发出了许多熵编码的方法来对信源数据进行压缩。其基本思想都是用可变长度的码字来表示出现概率不同的符号。用较短的码字表示出现概率较高的符号而用较长的码字表示出现概率较低的符号[28]。熵编码中两种比较重要的编码分别为霍夫曼编码和算术编码，而 H.264 中的基于上下文的自适应可变长编码和基于上下文的自适应二进制算术编码正是由这两种编码发展而来的。

3.2 基于上下文的自适应可变长编码

基于上下文的自适应可变长编码采用了霍夫曼编码的思想。霍夫曼编码为霍夫曼 1952 年提出的构造最佳码的编码方法。该方法完全依据字符出现概率来构造带权路径长度最小的二叉树从而实现平均码字长度最短，且各个符号编码的字头都是相异的，即任何一个符号的编码都不会成为另一个符号编码的前缀。CAVLC 主要用于对 H.264 中的 4×4 或 2×2 残差块变换量化系数经过 zig-zag 扫描后的数据的熵编码。CAVLC 利用了 4×4 变换块的如下几个特点优化熵编码设计：

1. 预测、变换和量化之后,块中会包含很多零系数，而且非零系数相对比

较分散，因此 CAVLC 利用 Run-Level 游程编码对连零的数据进行压缩。

2. 在 zigzag 扫描后，最高频的非零系数通常为±1，CAVLC 通过分别编码高频连续的±1(TrailingOnes)的个数和对应的符号对高频数据进行压缩。

3. 相邻块的非零系数的个数是相关的。因此，块中的非零系数个数是通过用相邻块的非零系数个数预测后查找表编码得到。实际编码的是非零系数个数的预测残差，冗余得到降低。

4. 扫描重排后的块系数的 level(幅度)在低频分量较大，往高频分量的 level 越来越小，CAVLC 利用了 Level 幅度的这种变化趋势和相邻 Level 的相关性，将相邻 Level 的幅度值引入当前 Level 编码中，也就是用相邻 Level 值预测下一个 Level 值的大小，其目的也是利用相关性减小冗余。

3.2.1 CAVLC 的编码过程

CAVLC 的编码流程如图 8 所示。具体编码过程为：

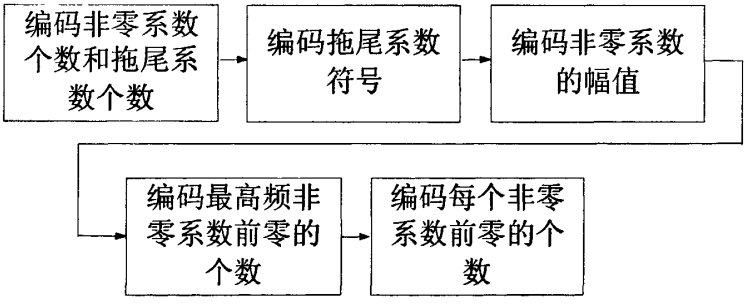

图 8　CAVLC 的编码过程图

Figure 8 The process of CAVLC encoding

1、对非零系数的总个数(TotalCoeffs)和拖尾系数个数(TrailingOnes)进行编码(Coeff token)。

拖尾系数指的是残差块中的±1。拖尾系数的个数应该小于等于 3 个，而若一个块中有多于 3 个的±1，则将最后 3 个作为拖尾系数，其它的按正常系数进行编码。

在编码每个块的 Coeff_token 时，有四个码表可供选择，包括一个定长码表和三个变长码表。四个码表分别针对不同的可能出现情况进行了不同的码字设定。表一针对参数个数比较少的情况进行设计，对较小的 TotalCoeffs 值赋予较短的码字；表三适合于参数个数比较多的情况，对较大的 TotalCoeffs 值赋予较短的码字；

表二介于表一与表三之间。表四则为 TotalCoeffs 和 TrailingOnes 赋予 6 比特定长码字。

码表的选择依赖于相邻已编码块的非零系数个数 N。这充分体现了 CAVLC 的上下文自适应性。如图 9 所示。当前编码块为 C 块，其相邻左边和上边的块分别为 B 块和 A 块。设其非零系数个数分别为 NB 和 NA. 当块 A、B、C 在同一条带时，NB 和 NA 都可用，此时 NC=round((NB+NA)/2)；否则，若 A、B、C 不再同一条带，当 NA 可用时，NC=NA；当 NB 可用时，NC=NB；若 NA 和 NB 都不可用，则 NC =0。

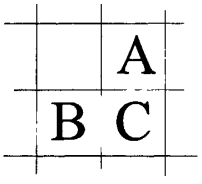

图 9　CAVLC 中相邻块的选择

Figure 9 The choice of neighbor blocks in CAVLC

当 NC 值确定后，码表的选择依 NC 值按表 3 进行。

表 3　Coeff_token 表的选择

Table 3　The choice of Coeff_token tables

NC 值	Coeff_token 表
0，1	表一
2，3	表二
4，5，6，7	表三
>=8	表四

2、对每个拖尾系数的符号进行编码

由于拖尾系数的幅值(level)恒为 1，所以只要对其符号进行编码即可。CAVLC 中用 0 代表正号，1 代表负号，按照倒序从高频到低频进行编码。

3、对除拖尾系数之外的每个非零系数进行编码

CAVLC 中除拖尾系数之外的每个码字幅值由两部分组成，分别为前缀(prefix)和后缀(suffix)。后缀的长度(suffixLength)是随着每个编码码字级别的幅度而自适应的变化的。这也体现了 CAVLC 的自适应性。较小的后缀长度适合于较低的幅度级别而较大的后缀长度则适合于较大的幅度级别。suffixLength 数值的初始化以及更新过程如下：

1) 将 suffixLength 初始化为 0，若非零系数的个数超过 10 个且拖尾系数少于 3 个，则将 suffixLength 初始化为 1。
2) 对最高频率的非零系数进行编码。
3) 根据非零系数的幅度增长 suffixLength 的值。如果这个系数的幅值大于表 4 中的一个预先设定的阈值，则 suffixLength 加 1。由表 4 可以看出，第一个阈值是 0，所以第一个系数编码结束后，suffixLength 的值总是增长。

表 4　增加 suffixLength 的阈值设定

Table 4　The thresholds for suffixLength

当前 suffixLength	阈值
0	0
1	3
2	6
3	12
4	24
5	48
6	无效

得到 suffixLength 的值之后，便可计算码字幅值的前缀和后缀了。具体计算过程如下：

1) 将有符号的 Level 转换成无符号的 levelCode：若 Level 为正，levelCode = (Level<<1) – 2；若 Level 为负，levelCode = - (Level<<1)–1；
2) 根据 levelCode 和 suffixLength 的值计算幅值前缀(level_prefix)：level_prefix = levelCode / (1<<suffixLength)；然后查表可得到对应的比特流；
3) 根据 levelCode 和 suffixLength 的值计算幅值后缀(level_suffix)：level_suffix = levelCode % (1<<suffixLength)；然后根据 suffixLength 的值确定后缀的长度。

4、对最后一个非零系数前零的总数(TotalZeros)进行编码。因为对于一个重排序后的块，其低频部分会出现多个连续的零系数，CAVLC 对此用一个 VLC 进行编码零的个数，这样这些连续为零的游程就不用编码了。

编码过程至此结束。

3.2.2　CAVLC 的解码过程

解码器端按照相反的顺序进行解码。其解码流程如图 10 所示。

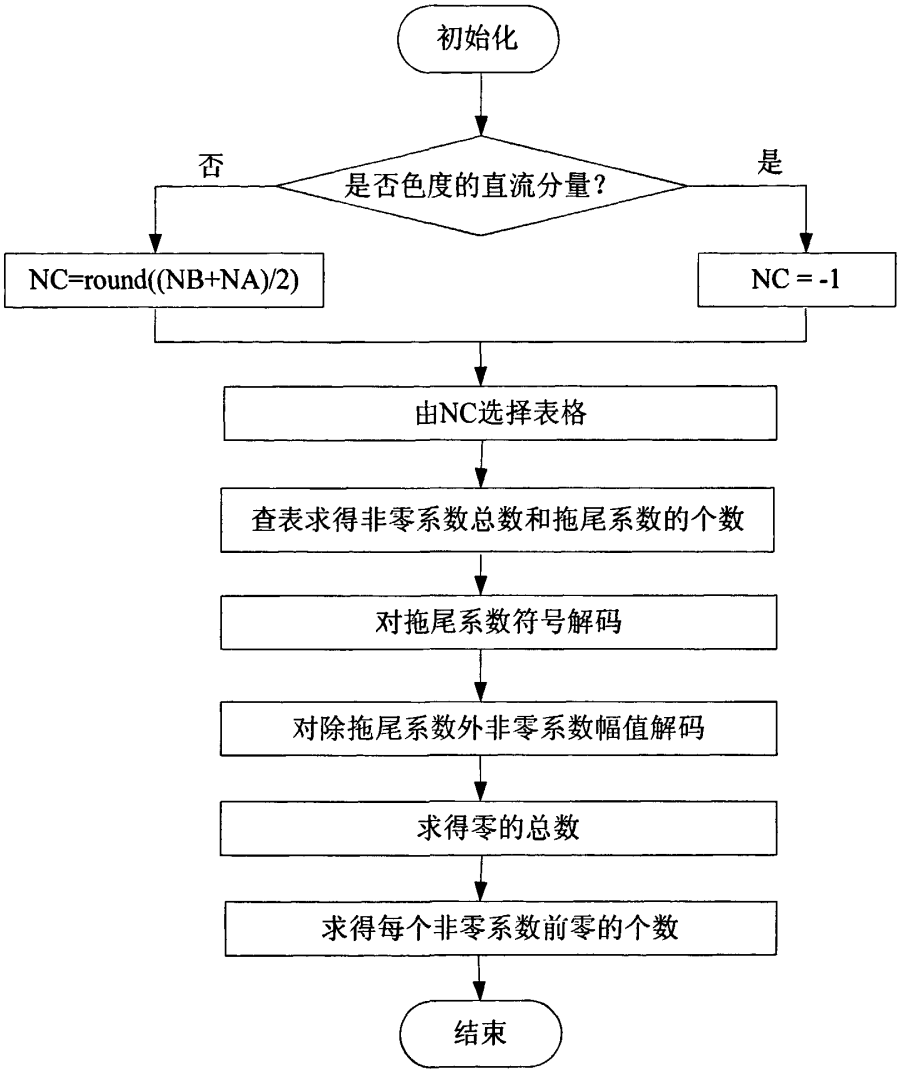

图 10　CAVLC 的解码过程图

Figure 10 The process of CAVLC decoding

首先在解码开始之前，解码器先要判断输入块的类型等参数来完成初始化工作。然后根据不同的块类型来进行不同的 NC 预测。得到 NC 值后，解码器将从码流中读入若干比特数据来进行 Coeff_token 的解码工作，完成 Coeff_token 的解码后，解码器根据得到的拖尾系数(TrailingOnes)的个数来从码流中读取相应长度的比特数据确定各个拖尾系数的符号。随后再进行幅值、总零个数、每个幅值前零的游程等解码工作。

3.3 CAVLC 解码过程的研究

3.3.1 JM 模型介绍

JM 模型[29]是基于 VC 平台的 H.264 编解码过程的实现模型，其编解码程序是经典的 C 程序。自其发布以来，已经推出多个版本，本文中将利用其中比较经典的一个版本 JM11.0 进行实验。

解压 JM11.0 压缩包后，共有若干个文件夹，其中 bin 文件夹中包含有各种编解码过程需要的配置文件，主要包括：

decoder.cfg：解码过程需要的配置文件，其中对解码过程中所需要的参数的值进行了设定；

encoder.cfg、encoder_baseline.cfg、encoder_extended.cfg、encoder_highquality_HD.cfg、encoder_main.cfg、encoder_yuv422.cfg、encoder_yuv444.cfg：这几个配置文件均为编码配置文件，包含了编码基本档次、扩展档次、主要档次、高清模式、YUV4:2:2 模式、以及 YUV4:4:4 等，并对各个不同档次编码时一些参数的初始值进行了设定，实验过程中如需要改变一些参数，可在此处更改。

ldecod 和 lencod 文件夹分别包含了解码过程和编码过程的 C 程序内容。实验过程如下，详细配置过程为：

1. 用 VC++6.0 打开源代码根目录下的工作区 tml.dsw；
2. 鼠标左键选中 ldecod 工程；
3. 打开 Project->Settings->Debug，在 Working directory 选项中填写./bin，在 Program arguments 选项中填写需要使用的解码配置文件，例如：decoder.cfg，然后确定修改；
4. 鼠标右键选中 ldecod 工程，选择鼠标右键菜单 Set as Active Project；
5. 编译运行解码器，完成解码。这个时候会在源代码根目录下的 bin 文件夹中生成几个新文件，其中 test_dec.yuv(对应为解码配置文件中的第三个参数)即为解码文件。

3.3.2 实验过程

本过程利用 JM11.0 模型，对 CAVLC 解码过程作了详细研究。对不同视频序列 CAVLC 解码过程中相关的一些参数做了统计。实验时电脑的配置为：

CPU：AMD Sempron(tm) Processor 3000+，1.61GHz 主频；

主板：昂达 C51MCP51；

内存： 1 G。

设定好参数值后，运行解码程序，得到如下结果，如图 11 所示，为解码视频测试序列 foreman.264 前三帧图像(分别为 I、P、B 帧)的解码过程信息，包括待解码视频序列、解码过程中 I、P、B 帧的量化参数 QP 以及解码耗时以及解码过程的总耗时等等。

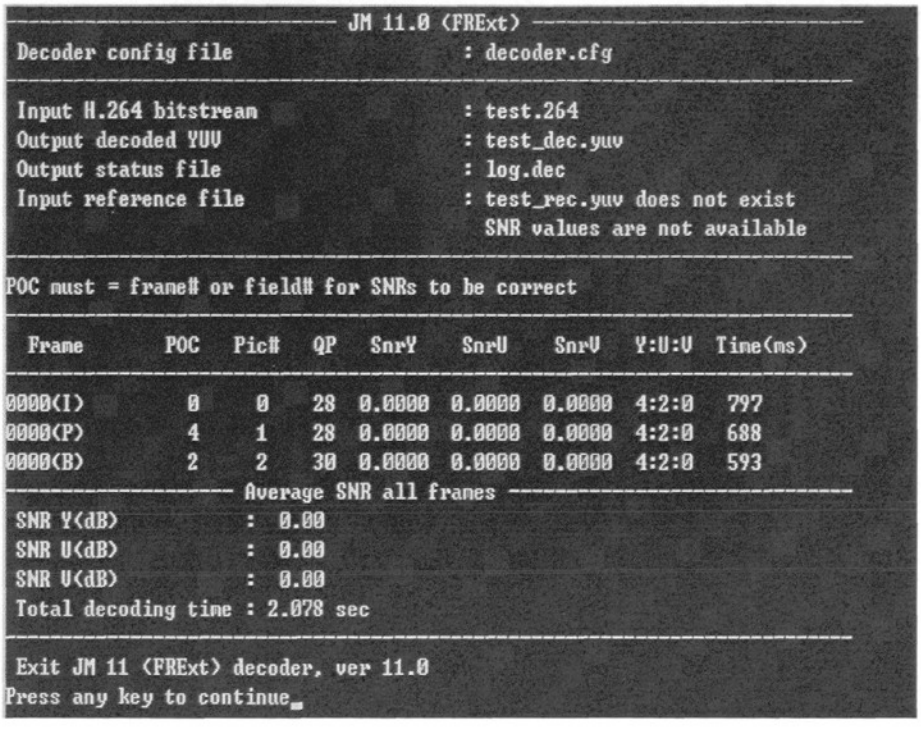

图 11　JM11.0 解码程序运行后的结果

Figure 11 The results after running the decoding program

为了更全面了解 H.264 中 CAVLC 编解码过程的详细情况，本文利用 JM11.0 模型对 CAVLC 编解码过程中的一些参数做了统计。本实验中选取了 5 个经典的视频测试序列 salesman、foreman、america、akiyo、claire 进行 H.264 编码。编码时设定了编码帧数为 9，其中第一帧 I 帧为 IDR 参考图像。编码视频的宽度和高度分别为 176 和 144，I 和 P 帧的量化参数选择 28，B 帧的量化参数选择 30。编码结果如图 12 所示。

```
Setting Default Parameters...
Parsing Configfile encoder.cfg.....................................................
------------------------------------------------------------------------------
----------------------------------------------------------

------------------------------ JM 11.0 (FREXt) -------------------------------

 Input YUV file                    : claire.yuv
 Output H.264 bitstream            : test.264
 Output YUV file                   : test_rec.yuv
 YUV Format                        : YUV 4:2:0
 Frames to be encoded I-P/B        : 5/4
 PicInterlace / MbInterlace        : 0/0
 Transform8x8Mode                  : 1

-------------------------------------------------------------------------------
  Frame   Bit/pic     QP    SnrY     SnrU     SnrV    Time(ms) MET(ms) Frm/Fld Ref
-------------------------------------------------------------------------------
0000(NVB)      176
0000(IDR)    14296   28   41.080   40.515   42.343        750        0     FRM   1
0002(P)       2528   28   40.440   40.274   42.090       1671      620     FRM   1
0001(B)        448   30   40.786   40.331   42.120       3094     2060     FRM   0
0004(P)       1992   28   40.439   40.266   42.036       2312     1387     FRM   1
0003(B)        328   30   40.390   40.392   41.954       3718     2662     FRM   0
0006(P)       2040   28   40.373   40.317   41.961       3063     2183     FRM   1
0005(B)        336   30   40.100   40.226   41.971       4391     3310     FRM   0
0008(P)       2096   28   40.344   40.058   41.988       3765     2752     FRM   1
0007(B)        416   30   40.304   40.054   41.932       5141     4092     FRM   0
-------------------------------------------------------------------------------
 Total Frames:  9 (5)
 LeakyBucketRate File does not exist. Using rate calculated from avg. rate
 Number Leaky Buckets: 8
     Rmin      Bmin      Fmin
    40800     14296     14296
    51000     14296     14296
    61200     14296     14296
    71400     14296     14296
    81600     14296     14296
    91800     14296     14296
   102000     14296     14296
   112200     14296     14296
-------------------------------------------------------------------------------
 Freq. for encoded bitstream       : 15
 Hadamard transform                : Used
 Image format                      : 176x144
 Error robustness                  : Off
 Search range                      : 16
 Total number of references        : 5
 References for P slices           : 5
 List0 references for B slices     : 5
 List1 references for B slices     : 1
 Total encoding time for the seq.  : 27.905 sec (0.32 fps)
 Total ME time for sequence        : 19.066 sec
 Sequence type                     : I-B-P-B-P (QP: I 28, P 28, B 30)
 Entropy coding method             : CAVLC
 Profile/Level IDC                 : (100,40)
 Motion Estimation Scheme          : Full Search
 Search range restrictions         : none
 RD-optimized mode decision        : used
 Data Partitioning Mode            : 1 partition
 Output File Format                : H.264 Bit Stream File Format
 Residue Color Transform           : not used
------------------- Average data all frames ----------------------------------
 PSNR Y(dB)                        : 40.47
 PSNR U(dB)                        : 40.27
 PSNR V(dB)                        : 42.04
 cSNR Y(dB)                        : 40.46 ( 5.84)
 cSNR U(dB)                        : 40.27 ( 6.11)
 cSNR V(dB)                        : 42.04 ( 4.06)
 Total bits                        : 24656 (I 14296, P 8656, B 1528 NVB 176)
 Bit rate (kbit/s)  @ 30.00 Hz     : 82.19
 Bits to avoid Startcode Emulation : 0
 Bits for parameter sets           : 176
-------------------------------------------------------------------------------
Exit JM 11 (FREXt) encoder ver 11.0
Press any key to continue_
```

图 12　claire 视频序列的编码结果

Figure 12 The encoding results of claire

上下文信息 NC 值

将编码后生成的 H.264 格式图像送入解码器进行解码，在解码器的程序中作者加入了部分代码来提取 NC 值的信息，并依据 CAVLC 解码过程中解码非零总数(TotalCoeffs)和拖尾系数(TrailingOnes)时参考的 Coeff_token 表对各个 NC 值的个数进行了统计，结果如表 5 所示(只针对解码亮度及色度交流系数时的 NC 值)，为了能够更直观，其折线图如图 13 所示。

表 5　NC 系数值统计结果

Table 5 Statistical results of NC

NC / 视频	0≤NC<2	2≤NC<4	4≤NC<8	NC≥8
foreman	2138	1058	378	41
salesman	1077	660	630	69
america	973	302	134	5
akiyo	919	365	249	46
claire	1089	442	152	28

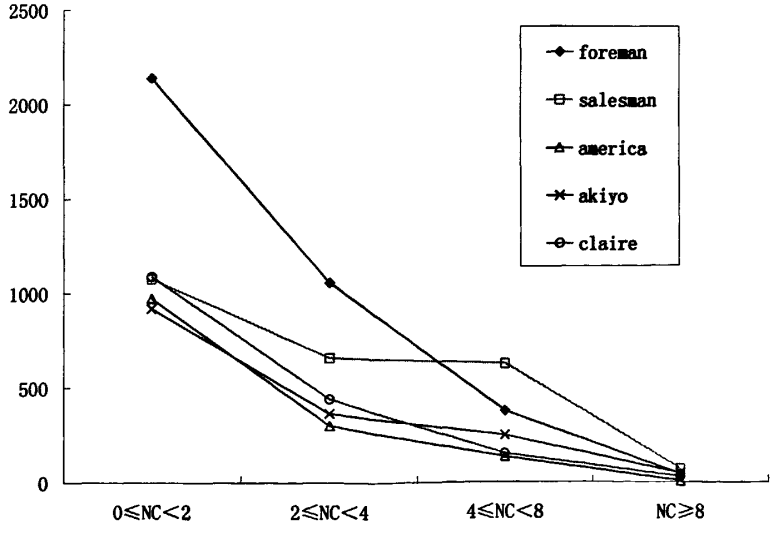

图 13　NC 系数值统计结果

Figure 13 Statistical results of NC

由表 5 和图 13 可知，对于不同的视频序列，其编码宏块的个数并不一样，这是由于 H.264 在编码过程中，对于某些宏块可以选择 skip 模式，即对于某些宏块，

在编码过程中可以跳过不编码。对于一个视频序列，其内容的运动复杂度影响着宏块编码的数目，内容运动越复杂，则其相邻块之间的差异也就越大，经运动补偿后块的残差系数也就越大，则可以跳过不编码的块也就越少；反之，内容运动越简单，其相邻块之间的差异也就越小，经运动补偿后块的残差系数也就越小，则可以跳过不编码的块也就越多。表 5 中视频序列 america 较之于其它视频序列不同的 NC 值编码宏块数均较少，这说明 america 序列中内容的运动复杂度较小，这与实际情况相一致。

另外，由表 5 和图 13 我们还可以发现，不论何种视频序列，其编码过程中 NC≥8 的个数远远小于 NC＜8 的个数。当 NC≥8 时，Coeff_token 的编码为定长码 6 比特，在解码的时候只要一次性读入 6 比特即可，省去了反复查表的过程，较之于 NC＜8 时的解码过程要简单得多。如果 NC≥8 的情况较少，则在解码 Coeff_token 就需要对码流中的比特逐位进行读入，并不断查找 Coeff_token 码表进行对比，直到找到匹配的码字为止。这是一个相当费时和消耗内存的过程，严重影响了 H.264 解码的实时性。同时由表 5 和图 13 还可以看到，解码 Coeff_token 时，0≤NC＜2 的情况要明显高于其它取值情况，而在 Coeff_token 码表中，0≤NC＜2 时对应的码字平均长度要高于其他情况，这也增加了 Coeff_token 解码的复杂性。

解码 Coeff token 时读入比特数

将编码后生成的 H.264 格式图像送入解码器进行解码，在解码器的程序中作者加入了部分代码来提取解码 Coeff token 时读入比特数的信息，并依据读入比特的个数进行了统计，结果如表 6 所示，为了能够更直观，其折线图如图 14 所示。

表 6　解码 Coeff_token 时读入比特数统计结果

Table 6 Statistical results of bits used to decode Coeff_token

比特数 / 视频	读入比特数≤3	3<读入比特数≤9	读入比特数＞9
salesman	1263	1139	34
foreman	2364	1218	33
america	1061	346	7
akiyo	997	556	26
claire	1223	460	28

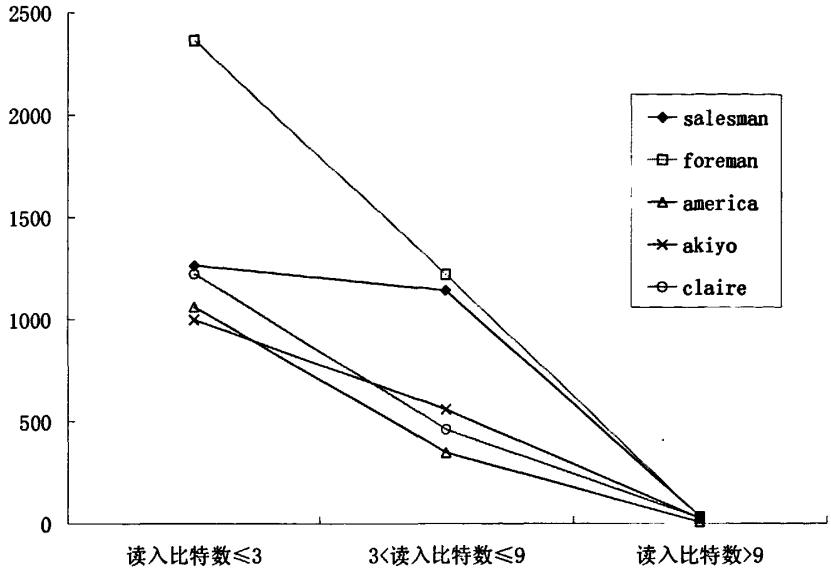

图 14　解码 Coeff_token 时读入比特数统计结果

Figure 14 Statistical results of bits used to decode Coeff_token

由表 6 和图 14 可知，在 CAVLC 解码 Coeff_token 过程中，对于这五种不同的视频序列，读入比特数的变化趋势是相同的，都是随着读入比特数的增加其出现次数逐渐减少。最终读入的比特数并不均衡，读入比特数小于等于 3 的情况最多，而读入的比特数大于 9 的情况在各个视频序列都比较少。这符合 CAVLC 编码的初衷，即对于出现概率较高的符号赋予较短的码字，而对于出现概率较低的符号则赋予较长的码字。同时，由于 Coeff_token 码表除 NC≥8 时是 6 比特定长码字外，其余码字都是不定长度的，这就给解码过程造成了一定的困难，因为在解码过程中读入的都是连续的比特流，解码器预先并不知道共有多少比特是表示 Coeff_token 的，因而在解码的时候只有逐个比特的读入数据并与码表进行对比来确定 Coeff token 的值，这对于 H.264 解码的实时性有着非常大的影响，因为 Coeff_token 码表的数据量非常的庞大！观察图 14 可知，相比五种不同的视频序列，在 foreman 序列中读入比特数小于等于 3 所占的比重要大于其它视频序列，这说明 foreman 序列的 CAVLC Coeff_token 解码效率更高。

3.4 CAVLC 解码过程的优化

参考 JM11.0 的 H.264 解码模型可以发现，CAVLC 参数的获得主要是利用了基于查表的方法来实现的。因为霍夫曼编码本身就是将不同的参数编码成不同长度的码字实现的。出现概率较大的符号用长度较短的码字来编码，而出现概率较

小的符号则用长度较小的码字来编码。在 CAVLC 中，这些符号的编码码字都储存在不同的码表中。以 Coeff_token 的解码过程为例：

Coeff_token 的编码码字的部分储存码表如表 7 所示。

表 7　Coeff_token 编码码字的部分储存码表

Table 7 Partial codes for encoding Coeff_token

拖尾系数个数	非零系数个数	0≤NC<2	2≤NC<4	4≤NC<8	NC≥8	NC= -1	NC =-2
0	0	1	11	1111	0000 11	01	1
0	1	0001 01	0010 11	0011 11	0000 00	0001 11	0001 111
1	1	01	10	1110	0000 01	1	01
0	2	0000 0111	0001 11	0010 11	0001 00	0001 00	0001 110
1	2	0001 00	0011 1	0111 1	0001 01	0001 10	0001 101
…	…	…	…	…	…	…	…

在解码过程中，解码器首先从解码比特流中读入 1 比特码字，然后查找码字储存表，如果表中有相同的码字，则停止读入码字，解出 TotalCoeffs 和 TrailingOnes，否则，解码器再读入 1 比特，用此 2 比特的码字去与储存表中的码字进行对比看有无相同的码字。若没有，则继续读入码字进行查找。如此循环往复，直到找到合适的码字。解码其它语法元素则与此类似。

由上面的介绍不难发现，在 CAVLC 解码过程中语法元素的查表查找过程是一个相当费时的过程。这严重影响了 H.264 的解码速度。除 NC=8 时，解码器可以一次性读入 6 比特外，其余情况下解码器必须按比特逐位读取，因为码字长度是不固定的。以 NC=0 为例，假设需要解码的码字为 0000000111，则解码器首先读进 1 比特 0，然后遍历储存表进行搜索，在搜索不到结果后，再读入 1 比特 0，再去搜索，直到读完 0000000111，才在储存表中找到对应的码字，然后解出 TotalCoeffs=4、TrailingOnes=0。整个解码过程进行了 10 次对储存表的搜索，而储存表中的数据量是相当大的，且码字的匹配过程中需要大量的读取和比较计算工作，因而此方法解码的时效性很差，不利于实时解码。

针对此种情况，本文对 Coeff_token 码表进行了分组优化。即将原来的变长码分成不同的几类，分别用不同的码表去储存。具体的分组码表如表 8、 表 9、表 10 所示(以 0≤NC<2 为例，其它类似)。

表 8　分组后前三比特不为零的 Coeff token 码字

Table 8 The first three non-zero codes after grouping

拖尾系数个数	非零系数个数	码字
0	0	1
1	1	01
2	2	001

表 9　分组后前三比特为零前九比特不为零的 Coeff_token 码字

Table 9 The first nine non-zero but first three zero codes after grouping

拖尾系数个数	非零系数个数	码字	拖尾系数个数	非零系数个数	码字
0	1	0001 01	3	4	0000 11
0	2	0000 0111	0	5	0000 0000 111
1	2	0001 00	1	5	0000 0001 10
1	6	0000 0000 110	2	5	0000 0010 1
2	6	0000 0001 01	3	5	0000 100
3	6	0000 0100	2	7	0000 0000 101
3	3	0001 1	3	7	0000 0010 0
0	4	0000 0001 11	3	8	0000 0001 00
1	4	0000 0011 0	3	9	0000 0000 100
2	4	0000 0101			

表 10　剩余码字

Table 10 The remaining codes

拖尾系数个数	非零系数个数	码字	拖尾系数个数	非零系数个数	码字
0	6	0000 0000 0111 1	2	12	0000 0000 0001 101
0	7	0000 0000 0101 1	3	12	0000 0000 0010 00
1	7	0000 0000 0111 0	0	13	0000 0000 0000 1111
0	8	0000 0000 0100 0	1	13	0000 0000 0000 001
1	8	0000 0000 0101 0	2	13	0000 0000 0001 001
2	8	0000 0000 0110 1	3	13	0000 0000 0001 100
0	9	0000 0000 0011 11	0	14	0000 0000 0000 1011
1	9	0000 0000 0011 10	1	14	0000 0000 0000 1110

表 10　剩余码字(续)

Table 10 The remaining codes(continued)

拖尾系数个数	非零系数个数	码字	拖尾系数个数	非零系数个数	码字
2	9	0000 0000 0100 1	2	14	0000 0000 0000 1101
0	10	0000 0000 0010 11	3	14	0000 0000 0001 000
1	10	0000 0000 0010 10	0	15	0000 0000 0000 0111
2	10	0000 0000 0011 01	1	15	0000 0000 0000 1010
3	10	0000 0000 0110 0	2	15	0000 0000 0000 1001
0	11	0000 0000 0001 111	3	15	0000 0000 0000 1100
1	11	0000 0000 0001 110	0	16	0000 0000 0000 0100
2	11	0000 0000 0010 01	1	16	0000 0000 0000 0110
3	11	0000 0000 0011 00	2	16	0000 0000 0000 0101
0	12	0000 0000 0001 011	3	16	0000 0000 0000 1000
1	12	0000 0000 0001 010			

将码字中前三比特不为零的码字放在同一个表格中，然后将前九比特不为零但前三比特为零的数据放在第二个表中，最后将剩余码字放在第三个表中。这样就将一个大的码表拆成三个小表，码字搜索匹配的过程将在这三个小表中进行，这大大缩小了搜索范围，减小了搜索的次数。具体解码流程如图 15 所示。

具体解码步骤为(仍以 0≤NC＜2 为例，其它类似)：

1. 通过计算得出的 NC 值找到匹配的码表；
2. 从码流中读 3 比特数据，若此 3 比特不为零，则选择码表 8 解码 TotalCoeffs 和 TrailingOnes，然后进行步骤 4；若此 3 比特为零，则进行步骤 3；
3. 从码流中读 6 比特数据，若此 6 比特不为零，则选择码表 9 解码 TotalCoeffs 和 TrailingOnes；若此 6 比特为零，则选择码表 10 解码 TotalCoeffs 和 TrailingOnes。进行步骤 4；
4. 继续解码其它语法元素。

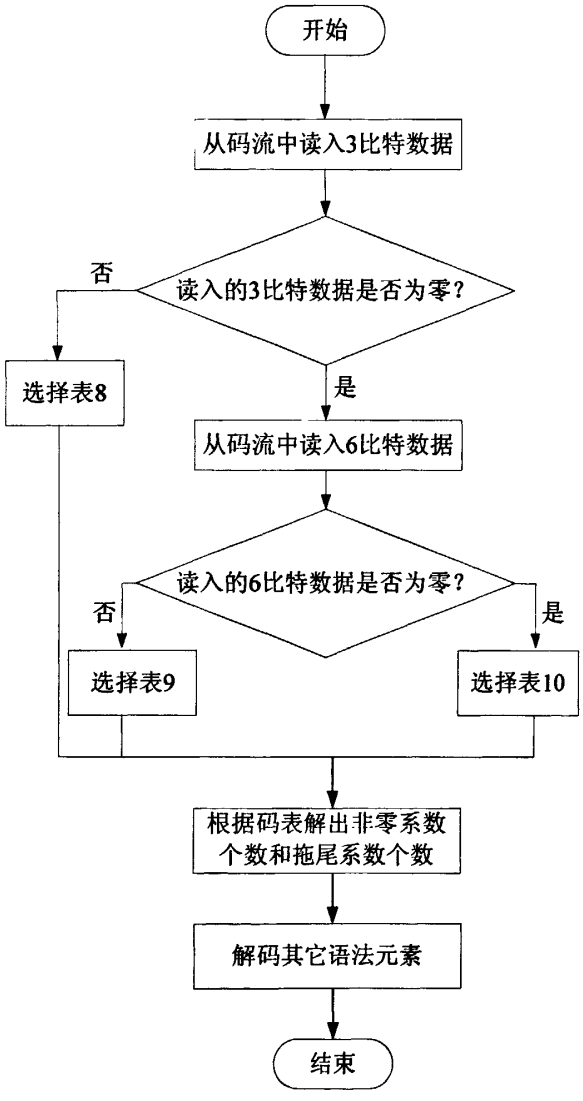

图 15 改进的 CAVLC 解码流程图

Figure 15 Process of improved CAVLC decoding method

3.5 实验过程及结果分析

基于以上分析，作者对优化后的算法编写了代码并与原 JM11.0 代码进行了解码对比,实验电脑的配置同 3.3.2 小节，作者选取了 5 种经典的视频测试序列

salesman、foreman、america、akiyo、claire，在编码时熵编码模式 entropy_coding_flag 均设为 0(1 代表 CABAC 编码)。其中的参数设置为：编码帧数为 9，其中第一帧 I 帧为 IDR 参考图像。编码视频的宽度和高度分别为 176 和 144，I 和 P 帧的量化参数选择 28，B 帧的量化参数选择 30。然后对得到的 H.264 视频文件进行解码时间测试，测试结果如表 11 所示，为了更直观，其柱形统计图如图 16 所示。

表 11　优化算法与原算法解码时间(单位：秒)

Table 11 Decoding time of optimization algorithm and original algorithm(Seconds)

	原算法	优化算法	优化算法/原算法
salesman	3.828	3.781	98.77%
foreman	3.890	3.859	99.20%
america	3.813	3.797	99.58%
akiyo	3.859	3.812	98.78%
claire	3.844	3.813	99.19%

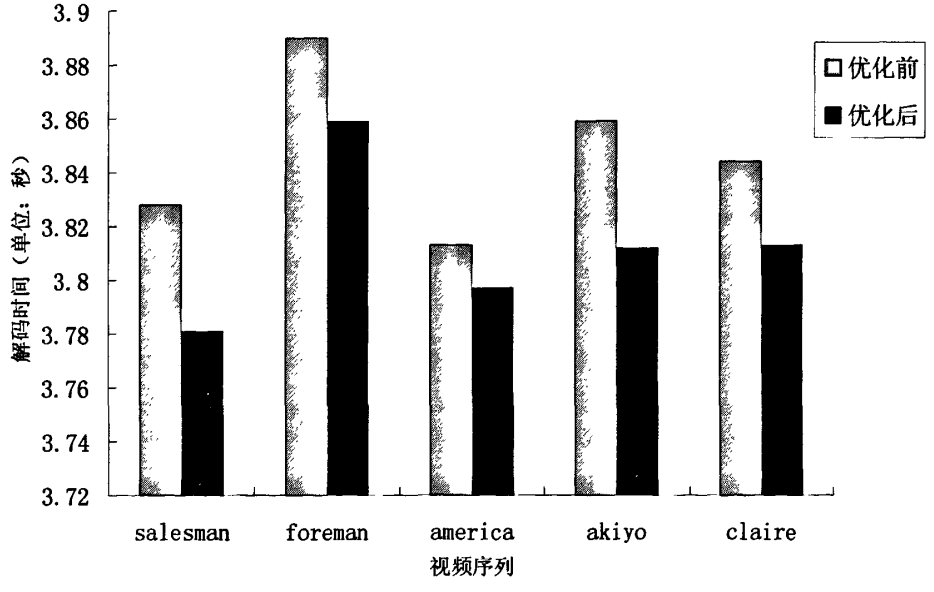

图 16 优化算法与原算法解码时间

Figure 16 Decoding time of optimization algorithm and original algorithm

由表 11 及图 16 可知，对于不同的视频序列，优化算法与原算法相比在解码时间方面均有不同程度的缩减，这是由于优化算法中数据的读取发生了变化，不再是逐比特读取，而是一次性读入数比特进行判断来选择相应码表，这样较之于

原算法节省了部分时间。而且读入相应比特数据后进行查表时，表格的大小发生了变化，解码器可以在分组后数据量较小的表格中进行查找，这样减少了查表时数据对比次数，从而节省了部分时间。实验数据表明优化后的解码算法较原算法在解码时间方面具有更好的性能，这对 H.264 在实时性要求较高场合的应用是十分重要的。

3.6 本章小结

本章对基于上下文的自适应可变长编码算法的核心思想、编解码原理以及编解码过程做了较为详细的阐述，对解码过程中的问题进行了一定的剖析，总结了其在解码过程中存在的一些缺点，然后利用 JM11.0 模型对其中的一些参数进行了统计并根据统计结果进行了分析，最后对解码过程进行了优化，使得 CAVLC 解码实时性得到了提高。

4 CABAC 编码研究及编码优化

4.1 CABAC 编码概述

H.264 中的另一种熵编码是基于上下文的自适应二进制算术编码，当语法元素 entropy_coding_flag 的值为 1 时，熵编码采用此种编码模式。与基于上下文的自适应可变长编码相比，CABAC 对视频数据的压缩效率更高[30]，然而其算法复杂度也较 CAVLC 复杂。CABAC 编码的基本思想源于算术编码。算术编码是上世纪 60 年代提出的一种基于香农信息论的编码方法，到 1976 年，其相关技术的介绍逐渐出现。与相关的其它编码方法相比，算术编码并不是对每一个符号去进行编码，而是对一个符号串去进行整体编码而得到一个最终编码结果。算术编码的基本思想是这样的[31]：首先给定[0，1)区间，然后依据各个符号的出现概率将[0，1)区间划分为不同的子区间。当编码第一个符号时，先选定该符号所在的区间并作为新的编码区间，编码下一符号时，再将此区间按概率划分，以此类推。最后得到一个最终区间。然后在该区间内选定一个比较容易表示的小数，其二进制值就是最终的编码输出。可见编码符号数越多，其输出小数位数也就越多，则最终二进制值也就越长。下面举例说明。

假设某信源可输出 a、b、c 三种不同的符号，且输出三种符号的概率为 p(a)=0.5，p(b)=p(c)=0.25。假设待编码序列为 abc,则编码过程如下：

1、确定各符号的区间，如图 17 所示。

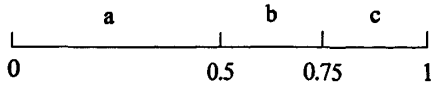

图 17 依概率确定的各编码符号区间

Figure 17 The code symbol interval according their probability

2、用 R 表示符号的编码区间，用 L、H 分别表示编码区间的下限和上限。则有：

$$L_1=L_0+R_0\times L \tag{9}$$

$$H_1=L_0+R_0\times H \tag{10}$$

3、当编码第一个符号a时，可确定其编码范围为：

$$L_1=L_0+R_0\times L=0 \tag{11}$$

$$H_1=L_0+R_0\times H=0.5 \tag{12}$$

4、当编码第二个符号b时，可确定其范围为：

$$L_2=L_1+R_1\times L=0.25 \tag{13}$$

$$H_2=L_1+R_1\times H=0.375 \tag{14}$$

5、当编码第三个符号c时，可确定其范围为：

$$L_3=L_2+R_2\times L=0.34375 \tag{15}$$

$$H_3=L_2+R_2\times H=0.375 \tag{16}$$

6、由此可得最终的编码区间为0.34375——0.375，为简便起见，可取0.36为最终编码输出，编码至此完毕。

解码过程：

1、当接收到0.36时，对照图17的各符号范围可知，第一个字符为a；

2、解出a后，减去a的范围下限并除以$p(a)$，得

$$(0.36-0)/0.5=0.72 \tag{17}$$

对照图17的各符号范围可知，第二个字符为b;

3、解出b后，减去b的范围下限并除以$p(b)$，得

$$(0.72-0.5)/0.25=0.88 \tag{18}$$

对照图17的各符号范围可知，第三个字符为c，则原符号序列为abc，解码至此结束。

4.2 CABAC编码过程

CABAC以算术编码的思想为基础，在编码过程中充分考虑了视频流的相关统计特性，使得H.264的压缩效率得到很大提高。相比算术编码，其主要改进的方面如下：

1. CABAC改进了算术编码中的多个信源概率符号，通过对待编码语法元素的二进制化使得最终编码符号只有0和1两个值，这样在编码的过程中只需要储存一个符号的概率就可以了，另一个符号的概率可由计算得到；
2. 由于对信源的符号概率统计比较困难，因而CABAC将每个符号可能出现的概率值进行预先计算并储存在一张表中，这样在实际编码过程中只要从表中选择一个比较接近的概率值进行替代即可；
3. 由于算术编码只有在最终编码完成才能输出码流，这使得信道在一定的时间段内会空闲而浪费信道资源，且随着编码区间的不断细化，最终的编码区间会非常小，这使得储存区间长度和区间下限等变量时需要极高的精度。CABAC为避免上述问题采用了区间规整的方法来不断输出码流并确保了区间在一定范围内。

CABAC的编码流程如图18所示，具体由三步组成：上下文建模、二值化、二进制算术编码。

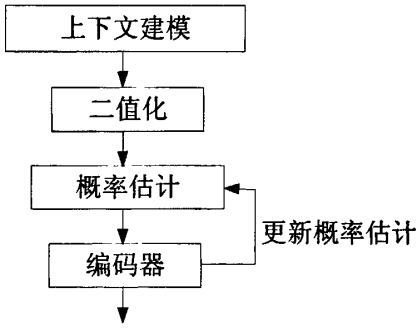

图18　CABAC编码流程图

Figure 18 The encoding process of CABAC encoding

4.2.1 上下文建模

与CAVLC类似，CABAC在编码过程中也充分利用了上下文信息之间的相关

性来减少数据的冗余度，实现对数据的压缩。CABAC 里有 399 个模型表，这些模型表通过变量 ctxIdx 索引，如表 12 所示。每个模型表中都列出了对应的 m、n 的值。同时 H.264 也为不同的语法元素分配了不同的上下文变量索引值。其中一些相对较为简单啊的语法元素被分配了较少的上下文变量而一些比较复杂的诸如残差数据等的语法元素被分配了较多的上下文变量。这也是 CABAC 编码的一个特点。

表 12　部分 ctxIdx 对应的 m、n 值

Table 12　Partial m、n for ctxIdx

CABAC_init_idc	初始化变量	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

在得到 m、n 的值后，就可以计算在初始化过程中的两个关键变量：概率状态索引(pStateIdx)和最大概率值(valMPS)。具体计算方法为：

$$preCtxState=Clip3(1,126,((m\times Clip3(0,51,SliceQP_Y))>>4)+n) \tag{19}$$

```
if(preCtxState<=63)
{
pStateIdx=63- preCtxState
valMPS=0
}
else
{
pStateIdx= preCtxState-64
valMPS=1
}
```

其中$SliceQP_Y$为条带数据的量化步长，$Clip3(x,y,z)$的含义是如果z在(x,y)的范围则直接取z，如果不在(x,y)范围则取边界值。对于P、SP和B条带类型，初始化过程也取决于CABAC_init_idc语法元素的值。CABAC_init_idc的值在条带头中解出。

得到概率状态索引后，便可以根据索引值确定概率值。在CABAC中共有64个有代表性的概率值来表示LPS（Least Probable symbol)的概率。这64个概率值为：

$$P_\sigma=\alpha\times P_{\sigma-1}, \tag{19}$$

$$P_0=0.5,\ \sigma=1,\ 2,\ \ldots,\ 63 \tag{20}$$

$$\alpha=(0.01875/0.5)^{1/63} \tag{21}$$

实际查找过程中由P_σ和$\bar{\omega}$（$\bar{\omega}$为0和1中概率较大的字符，即MPS(Most Probable Symbol))组成128个概率模型，最终根据各个语法元素的上下文变量计算出来的概率状态索引(pStateIdx)和最大概率值(valMPS)来选择一个模型。H.264会为每一个语法元素分配若干个上下文变量，这些上下文变量的选择有些是固定的，而有些是需要根据上下文信息进行选择的，这也体现了CABAC编码的自适应性。如果该语法元素不是残差数据，则上下文变量的选择主要依赖于相邻编码块的信息(一般为左边和上边的块)或前语法元素中已编码的二进制值。如果该语法元素为残差数据，则上下文变量的选择主要依赖于当前块中以编码数据的个数或当前编码语法元素在编码块的之字扫描中的位置。

4.2.2　二进制化

CABAC 编码不同于算术编码的一个特点就是 CABAC 中信源的输入符号只有 0 和 1 两种，这也使得编码过程不至于太复杂。因此编码过程中需要将各个语法元素用一串二进制比特表示出来，这就是语法元素的二进制化过程。CABAC 针对不同的语法元素设计了几种不同的二进制化方案，包括基本方案和串接方案。基本方案中包括四种码：一元码、截断一元码、K 阶指数哥伦布码和定长码，串接方案是由基本方案中不同的编码方式串接起来的。根据各种码的不同特点，不同的语法元素用不同的编码方式去表示。例如，定长码用于表示简单的语法元素而 K 阶指数哥伦布码则用于表示数值变化范围较大的残差数据。下边详述基本方案中四种码的编码方法：

1、一元码：此种编码方法将待编码的无符号整数转换成相应个数的“1”并加后

缀“0”，如表13所示。

表13 一元码编码后的二进制串

Table 13 The Binary string by Unary

语法元素值	二进制串
0 (I_NxN)	0
1	10
2	110
3	1110
4	11110
5	111110
…	…

2、截断一元码(又叫舍位一元码)：此种编码方法包含一个参数cMax，将待编码的数与cMax进行比较，若小于cMax，则按照一元码编码方法进行编码；否则，被转换成相应个数的“1”。

3、K阶指数哥伦布码：编解码的表达式为

$$M=\text{floor}\left(\log_2((code\_num)>>k+1)\right)+k \tag{22}$$

$$INFO=code\_num+2^{K}-2^{M} \tag{23}$$

4、定长码：此种编码方法也包含参数cMax，待编码整数编码后的统一码长为

$$length=\text{ceil}[\log_2(cMax)] \tag{24}$$

4.2.3 二进制算术编码

CABAC的编码过程与算术编码类似，不同的是：

1. CABAC编码的输入符号只有0和1两个值；
2. 区间范围的选择是通过查表而不是计算得到的，这样可以提高CABAC的编码速度。

在CABAC编码中有两个编码器可供选择，分别为规则编码器和旁路编码器。

当编码符号概率接近于 0.5 的时候，编码后概率不需要更新，此时选择旁路编码器可以提高编码速度，节省编码时间；当符号概率为一般情况的时候，选择规则编码器进行编码，编码完毕后需要对概率状态进行更新。编码流程如图 19 所示。

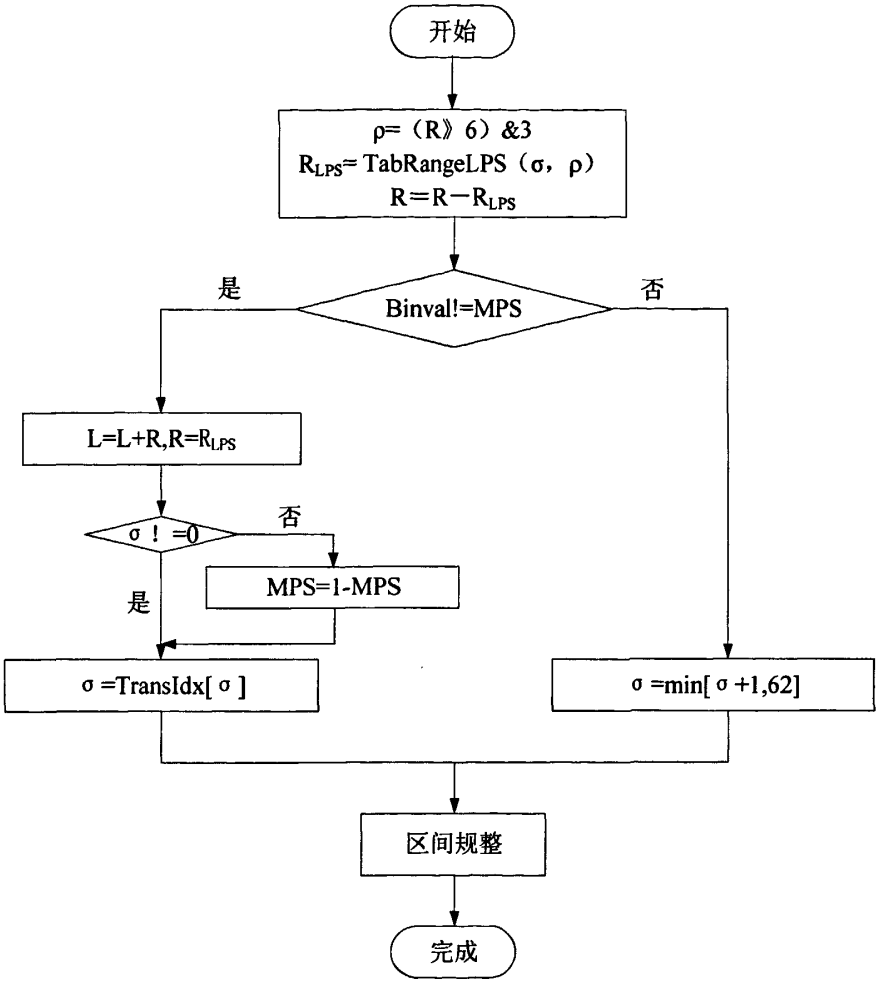

图 19　CABAC 编码过程

Figure 19 The process of CABAC encoding

具体编码过程为：

1、编码器首先把编码区间量化：编码区间 R 的范围为(28，29)，具体量化方法是通过 R 的第六位和第七位进行四等分，量化索引

$$\rho=(R\gg 6)\&3 \tag{25}$$

2、进行编码前的区间更新:在算术编码中区间更新是通过 $Rx=R\times Px$ 进行的，

这就需要进行乘法运算，降低了编码速度，影响了编码的实时性。CABAC 为了避免这个问题，采用了查表的方法来代替乘法运算，将预先计算好的结果储存于一张 64×4 的表中，具体编码的时候只要查找这个表就可以了，如表 14 所示。该表中列出了相应的概率状态索引以及编码区间的小概率状态(R_{LPS})的值。据表

$$R_{LPS}= TabRangeLPS(\sigma, \rho) \tag{26}$$

$$R_{MPS}=R-R_{LPS} \tag{27}$$

表 14 部分区间概率状态索引以及编码区间对应的 R_{LPS} 表

Table 14 Partial R_{LPS} table for some pStateIdx and encoding interval

pStateIdx	qCodIRangeIdx			
	0	1	2	3
0	128	176	208	240
1	128	167	197	227
2	128	158	187	216
3	123	150	178	205
4	116	142	169	195
5	111	135	160	185
…	…	…	…	…

3、通过输入的 binval 值来更新当前的编码区间：如果当前的 binval 值是 MPS 值，则编码区间的下限不变，只将当前编码区间长度更新为 R_{MPS}；如果当前的 binval 值是 LPS 值，则区间下限变为 R_{LPS}，且编码区间变为 R_{LPS}。由此可知，编码 MPS 的时候只是编码区间长度发生了变化，而编码下限却并没有改变。而影响编码输出值的正是编码下限，因此如果编码比特流中出现大量 MPS 值或者 MPS 值相对较多的时候，编码过程将相对比较简单且压缩效果比较高，更接近于信源的熵。

4、概率状态的更新：在 CABAC 中，每编码完一位都要进行概率状态更新。这样可以更好地接近信源符号的实际概率。具体的更新方法为：每次输入的 binval 值为 MPS 时，表明输入 MPS 的概率增大，σ 加 1，当 σ 达到其最大值 62 时，其

值不再随着输入的 MPS 增大，直到出现 LPS，其值才会更新。当输入的 binval 值为 LPS 时，如果 σ 为 0，则表明此时 MPS 的概率和 LPS 的概率相等，MPS 和 LPS 进行互换，否则，σ 的值需要查状态转移表得到，如表 15 所示。

表 15　σ 值的状态转移表

Table 15　State transition table for σ

pStateIdx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
transIdxLPS	0	0	1	2	2	4	4	5	6	7	8	9	9	11	11	12
transIdxMPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
pStateIdx	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
transIdxLPS	13	13	15	15	16	16	18	18	19	19	21	21	22	22	23	24
transIdxMPS	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
pStateIdx	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
transIdxLPS	24	25	26	26	27	27	28	29	29	30	30	30	31	32	32	33
transIdxMPS	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
pStateIdx	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
transIdxLPS	33	33	34	34	35	35	35	36	36	36	37	37	37	38	38	63
transIdxMPS	49	50	51	52	53	54	55	56	57	58	59	60	61	62	62	63

5、对编码区间进行规整：CABAC 编码继承了算术编码的方法，对一个连续的比特串去进行编码并最终输出结果。这里会有两个问题产生：

1) 因为算术编码是对一个比特串连续编码的，并不是对每一个比特去进行单独编码，所以其输出是不连续的，在编码一个比特串的过程中并不输出，这就造成了信道资源的浪费，而且视频传输的实时性也会受到一定的影响；
2) 如果输入码流较长，随着编码比特的递进，最终编码区间将会非常小，最终必须以极高的精度来记录编码区间 R 和编码区间下限 L。CABAC 采用了如下方法来解决这两个问题：在实际的编码过程中，随着输入比特的不

断增加，编码区间将变的越来越小，因此其编码区间上下限也越来越接近，当编码区间上下限的最高位相同时，则此位不可能再改变，此时可以将此位比特输出来避免信道的空闲。输出后同时将上限和下限左移一位，这相当于给编码区间的上下限同时扩大 2 倍，则编码区间相对于原来变大，避免了以极高的精度来记录编码区间 R 和编码区间下限 L 的问题。

4.3 CABAC 编码过程中存在的问题

较之于 CAVLC，CABAC 在数据压缩效率方面有一定的提高，但是其算法复杂度也随之提高，这使得 CABAC 在实际应用的时候受到一定的限制。CABAC 编码为了减少计算复杂度采取了一些措施，比如：

1. 将各符号的近似概率提前计算出来并以表格形式储存，在实际编码的时候用有限状态机来索引概率，这样避免了概率的实时计算，节省了编码时间；
2. 在算术编码过程中，编码每个符号时编码区间的范围是通过 R=R×L 实现的，这就涉及到乘法运算，在实际的计算机编码过程中，乘法的计算是很耗时的，尤其是涉及到浮点数乘法的时候。CABAC 在实际编码过程中也通过采用提前计算好并用表格储存的方式来避免这个问题，需要的时候只要查表即可完成。

但是 CABAC 中仍然存在着一些缺点,例如:

1. CABAC 在编码过程进行之前的初始化过程中假定符号“0”和符号“1”的概率相等(即 P_0=0.5)，这没有很好的利用先验知识，会对数据压缩效率造成一定的影响；
2. CABAC 在每编码完 1 比特后都要进行概率状态的更新，这增大了 CABAC 编码的复杂度，且对 H.264 的实时性影响较大，不利于实时编码传输；
3. 如果待编码语法元素二进制化后的比特较长，则在编码过程中，随着比特输入个数的增加，编码区间将变得越来越小，CABAC 在编码过程中不得不随时进行编码区间的规整来克服这个问题，这也增加了 CABAC 复杂性，影响了其实时性。

4.4 CABAC 编码过程优化

由于 CABAC 编码存在着诸如 4.3 小节所述的诸多缺点，本节拟对 I 条带中宏

块类型(mb_type)的 CABAC 编码过程做一定的改进。根据 H.264 标准，I 条带中宏块类型二进制化后的值如表 16 所示。观察表 16，我们可以发现如下特点：

1. 在所有类型的宏块中，只有 4x4 块和 PCM 宏块二进制化后结果比较短，其余的宏块类型二进制化后结果均相对较长；
2. 除 I_4x4 块和 I_PCM 块外，其余类型宏块二进制后前两个比特值均为“0”和“1”。

表 16　I 条带中语法元素 mb_type 的二进制化结果

Table 16　Binary results of mb_type in I slice

mb_type 的值 (名称)	二进制码串	mb_type 的值 (名称)	二进制码串
0 (I_4x4)	0	13 (I_16x16_0_0_1)	101000
1 (I_16x16_0_0_0)	100000	14 (I_16x16_1_0_1)	101001
2 (I_16x16_1_0_0)	100001	15 (I_16x16_2_0_1)	101010
3 (I_16x16_2_0_0)	100010	16 (I_16x16_3_0_1)	101011
4 (I_16x16_3_0_0)	100011	17 (I_16x16_0_1_1)	1011000
5 (I_16x16_0_1_0)	1001000	18 (I_16x16_1_1_1)	1011001
6 (I_16x16_1_1_0)	1001001	19 (I_16x16_2_1_1)	1011010
7 (I_16x16_2_1_0)	1001010	20 (I_16x16_3_1_1)	1011011
8 (I_16x16_3_1_0)	1001011	21 (I_16x16_0_2_1)	1011100
9 (I_16x16_0_2_0)	1001100	22 (I_16x16_1_2_1)	1011101
10 (I_16x16_1_2_0)	1001101	23 (I_16x16_2_2_1)	1011110

表 16　I 条带中语法元素 mb_type 的二进制化结果(续)

Table 16　Binary results of mb_type in I slice(Continued)

mb_type 的值 (名称)	二进制码串	mb_type 的值 (名称)	二进制码串
11 (I_16x16_2_2_0)	1001110	24 (I_16x16_3_2_1)	1011111
12 (I_16x16_3_2_0)	1001111	25 (I_PCM)	11

由此，我们可以对 I 条带中宏块类型(mb_type)的 CABAC 编码作如下改进：在编码前首先判断宏块类型，如果宏块类型为 I_4x4 块或 I_PCM 块，则直接对其按位进行编码；否则，如果宏块类型不为 I_4x4 和 I_PCM，则编码时跳过前两个比特，直接对余下比特进行编码。改进后的 CABAC 过程如图 20 所示。

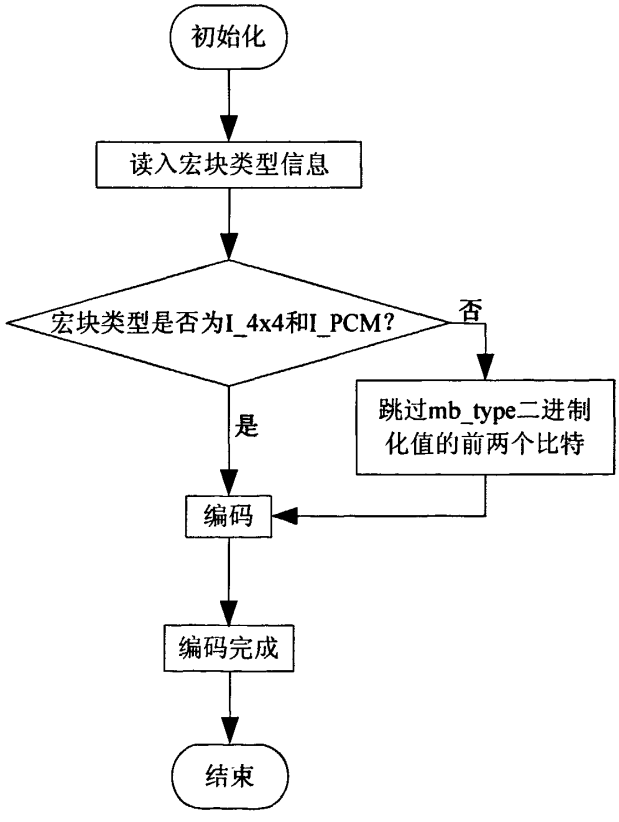

图 20　改进后的 I 条带中宏块类型的 CABAC 编码过程

Figure 20 The process of improved CABAC encoding for mb_type in I slice

具体编码过程为：

1. 初始化编码引擎，准备编码；
2. 对输入的宏块类型信息进行判断：如果宏块类型为 I_4x4 或 I_PCM，跳至步骤 4，否则，跳至步骤 3；
3. 跳过宏块类型二进制化值的前两个比特，跳至步骤 4；
4. 对比特串进行编码。

4.5 实验过程及结果分析

基于 4.4 小节所做的改进,本节对改进后的编码过程利用 JM11.0 模型进行了实验。实验电脑的配置同 3.3.2 节。仔细分析 JM11.0 模型中关于宏块类型的 CABAC 编码过程后可知，宏块类型 mb_type 的 CABAC 编码主要是利用函数 writeMB_typeInfo_CABAC 实现的，而在 writeMB_typeInfo_CABAC 函数中又多次调用了 biari_encode_symbol 函数和 biari_encode_symbol_final 函数对二进制化后的每一个比特位进行熵编码。作者选取 5 种经典视频测试序列 salesman、foreman、america、akiyo、claire，对原有方法中和改进后的 I 条带中宏块类型的编码方法中 biari_encode_symbol 函数和 biari_encode_symbol_final 函数的调用次数做了统计，统计结果如表 17 所示，为了更直观，其柱形统计图如图 21 所示。其中的参数设置为：编码帧数为 9，其中第一帧 I 帧为 IDR 参考图像。编码视频的宽度和高度分别为 176 和 144，I 和 P 帧的量化参数选择 28，B 帧的量化参数选择 30。

表 17 编码 I 条带 mb_type 时函数 biari_encode_symbol 和 biari_encode_symbol_final 的调用次数

Table 17 Calls of biari_encode_symbol and biari_encode_symbol_final while encoding mb_type in I slice

算法 / 视频	优化前	优化后	优化后/优化前
salesman	7339	5896	80.34%
foreman	7208	5762	79.94%
america	7706	6114	79.34%
akiyo	7425	5928	79.84%
claire	7685	6038	78.57%

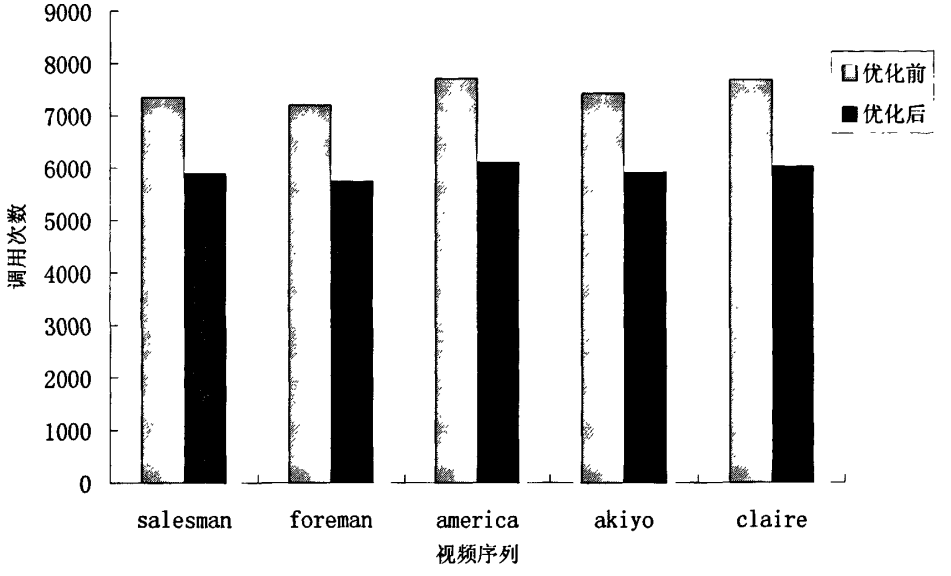

图 21 编码 I 条带 mb_type 时函数 biari_encode_symbol 和 biari_encode_symbol_final 的调用次数

Figure 21 Calls of biari_encode_symbol and biari_encode_symbol_final while encoding mb_type in I slice

由表 17 及图 21 可以发现，对于不同的视频序列，优化后的编码方法与原编码方法相比，对函数 biari_encode_symbol 和 biari_encode_symbol_final 的调用次数均减少了 20%左右。这是由于程序在对每一个语法元素进行二进制编码时，会调用数次相关的二进制编码函数，改进后的编码方法减少了待编码二进制比特串的长度，因而相关函数调用次数也得到了相应的减少。对于一个程序而言，调用函数时势必会增加耗时，增加其调用函数的次数，则其执行耗时也会相应增加，因而优化后的编码方法其编码过程较原有方法编码过程更省时，这在对实时性要求较高的场合是十分重要的。仍然采用相同的电脑配置、相同的视频序列以及编码参数设置，作者对优化后的编码方法与原有编码方法的编码时间进行了统计，如表 18 所示，为了更直观，其柱形统计图如图 22 所示。

表 18 编码不同视频序列的编码时间(单位：秒)

Table 18 The encoding time for different video sequences(Seconds)

算法 / 视频	优化前	优化后	优化后/优化前
salesman	23.689	22.844	96.43%
foreman	23.999	23.219	96.75%

表 18 编码不同视频序列的编码时间(单位：秒)(续)

Table 18 The encoding time for different video sequences(Seconds)(Continued)

算法 / 视频	优化前	优化后	优化后/优化前
america	22.925	22.218	96.92%
akiyo	22.735	22.141	97.39%
claire	22.671	22.109	97.52%

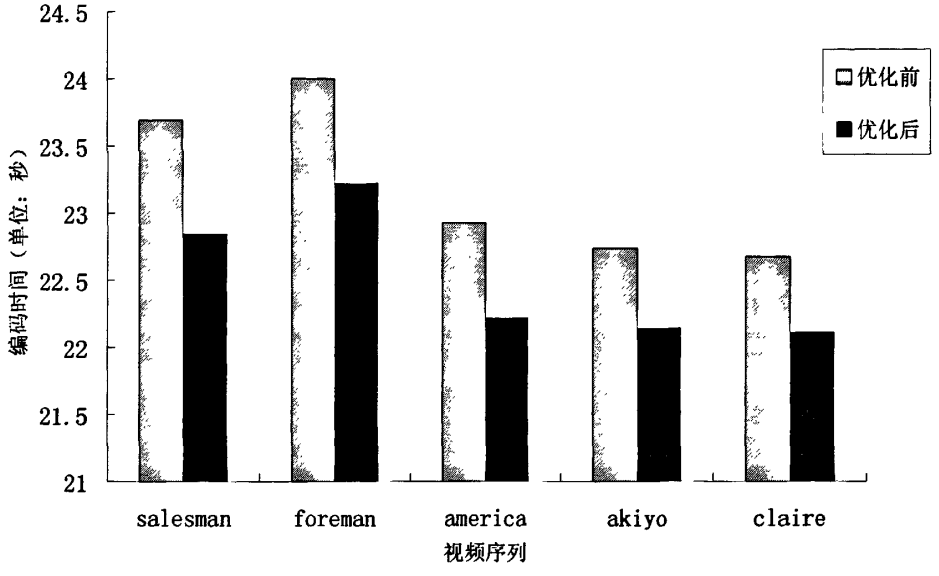

图 22 编码不同视频序列的编码时间(单位：秒)

Figure 22 The encoding time for different video sequences(Seconds)

由表 18 及图 22 可以发现，对于不同的视频序列，优化后的编码方法与原编码方法相比，其编码时间均有所减少。这是由于除相关解码函数调用次数的减少节省部分时间外，改进后的算法缩短了待编码比特串的长度，依据上下文信息选择概率模型次数以及编码完成后的区间规整次数均得到了缩减。选取的 5 种测试序列编码速度的提高均为 3%左右。

同时，根据对基于上下文的自适应二进制算术编码过程的介绍可知，减小待编码比特串的长度可以减小编码后比特串的输出长度，这在一定程度上也减小了编码后生成的 H.264 文件的体积，使得 H.264 视频文件的储存以及网络传输更容易。

4.6 本章小结

本章对基于上下文的自适应二进制算术编码算法的核心思想、编码原理以及编码过程做了较为详细的阐述，对编码过程中的问题进行了一定的剖析，总结了其在编码过程中存在的一些缺点，然后对编码 I 条带宏块类型时基于上下文的自适应二进制算术编码的编码过程进行了改进，并通过不同的视频测试序列利用 JM11.0 模型比较了改进前后 CABAC 的编码耗时，实验数据表明改进后的编码方法较原编码方法在实时性方面得到了一定的提高。

5 结论和展望

5.1 总结

随着科学技术的不断发展，人们对视频信息的需求量越来越大，而且对其品质的要求也越来越高，如何高效的对视频信息进行编解码处理以及传输成为当前多媒体通信领域研究的热点和重点，因此在近几年中，多种视频编码标准被不断地提出。而 H.264 作为一种最新的视频编码标准，在数据压缩效率以及网络传输能力方面有着很好的性能，当然其高性能的获得是以增加视频编码过程中的算法复杂度为代价的，相比 H.263，其编码复杂度增加了 2 倍左右，解码复杂度也增加了 1 倍左右，但是随着计算机技术以及芯片技术的飞速发展，高效的 CPU 处理器以及芯片不断地出现，H.264 在实际场合中已经得到了应用。

然而，在一些实际应用场合，H.264 的编解码实时性仍不太理想，于是本文在介绍了 H.264 视频标准的关键环节之后，对熵编解码中两种比较重要的编解码方法——基于上下文的自适应可变长编码和基于上下文的自适应二进制算术编码进行了研究与优化。这两种算法都是基于香农的信息论提出的，在编码的过程中均采用了一定的方法去逼近信源各符号概率的真实熵，并且还充分利用了上下文信息，使得数据压缩效率得到了进一步提高。相比较而言，基于上下文的自适应二进制算术编码由于利用了基于比特串整体编码的方法，数据压缩效率更高，当然其算法复杂度也较高，这也限制了其的应用。在 H.264 的基本档次和扩展档次中均没有利用 CABAC 编码。

论文的主要内容以及期间作者的主要工作包括：

1. 对 H.264 视频标准的整个编解码过程进行了简要的介绍，对其中的关键环节(尤其是帧内帧间预测、变换、量化和熵编码等环节)所涉及到的主要问题进行了阐释；
2. 在简要介绍 H.264 视频标准中关键环节的基础上，作者对基于上下文的自适应可变长编码和基于上下文的自适应二进制算术编码两种熵编码方法进行了分析与研究，总结了其优缺点；
3. 随后，作者对基于上下文的自适应可变长编码的解码过程和编码 I 条带宏块类型时基于上下文的自适应二进制算术编码的编码过程进行了一定的优化与改进，并通过 VC6.0 利用 JM11.0 模型对优化前后的算法进行了一定的性能测试，实验数据表明，优化后基于上下文的自适应可

变长编码的解码时间和编码 I 条带宏块类型时的基于上下文的自适应二进制算术编码的编码时间较原算法均有所缩短，这使得 H.264 的实时性得到了提高。

H.264 视频编码方法较之于以前的编码方法，在各个环节都有较大的改进，因而其对数据冗余度的压缩也相对较高。H.264 视频标准的出现把运动图像压缩技术以及视频通信技术推向了一个新的高潮，很多生产厂家都在致力于研究可支持 H.264 技术的视频编解码芯片，且有一些已经推向市场。

5.2 展望

H.264 的出现将基于块的混合编码方法推向了一个新的起点，其编码各环节中新提出的各种算法也为以后视频编码的发展起到了引导作用。随着科技的不断发展，H.264 技术必将在更为广阔的领域得到高效的运用。在 H.264 的熵编解码方面，除了本文所改进的方面外，在其它方面还可以进行一定的改进来提高其编解码效率。而且随着人们对熵编解码研究的不断深入，还可以提出新的基于香农信息论的熵编解码方法来进一步逼近信源符号熵，提高数据压缩效率。

同时，在除熵编码外的许多其它环节，例如帧内帧间预测、环路滤波等环节也可以进行改进来简化编解码算法，提高编解码实时性，这将是 H.264 编解码技术的一个研究方向。

H.264 技术的出现与发展不论对视频编解码研究的科研领域还是视频产品的应用市场都产生了较大的影响：

1. 在科研方面，H.264 充分的发展了基于块的混合编码方法。在此之前，不少专家和学者曾认为基于块的混合编码方法已经没有可发展的空间，而 H.264 技术的出现则证明了基于块的混合编码方法还有相当大的发展空间，对其的研究还应该继续；
2. 在视频产品市场，H.264 的出现对以往的视频产品产生了很大的冲击作用，很多视频产品不得不进行更新换代来支持新的视频编解码技术，而视频解码芯片的各个生产厂家也在争着推出新的芯片来抢占市场，因此 H.264 有着相当巨大的市场潜力。

参考文献

[1]T.Wiegand,G.J.Sullivan,G.Bjontegaard,and A.Luthra.Overview of the H.264/AVC video coding standard.Circuits and Systems for Video Technology.IEEE Transactions.Jul,2003.Vol.13,no.7.560～576.

[2]T.Wigand,X.Zhang,and B.Gird,Long-Term Memory Motion-Compensated Prediction Systems for Video Technology.IEEE Transactions.1999.no.9.70~84

[3]殷晓莹,龚建荣.视频压缩编码标准及其发展.信息安全与通信保密.2005.4.125～127

[4]王嵩等.H.264 视频编码标准.DIGITAL TV&DIGITAL VIDEO.2003.6.25～27

[5]G.K.Wallace,The JPEG still-picture compression standard.Communications of ACM.Apr,1991.Vol.34

[6]余兆明,查日勇,黄磊,周海骄.图像编码标准 H.264 技术.北京.人民邮电出版社.2006.1~3

[7]ISO/IEC JTC 1/SC 29.15444-5-2003.Information technology-JPEG 2000 image coding system: Reference software

[8]ISO/IEC JTC 1/SC 29.15444-1-2004.Information technology-JPEG 2000 image coding system: Core coding system

[9]ISO/IEC JTC1.11172-2-1993.Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s;part 2:video

[10]Chang Yung-Chi,Huang Chao-Chih,Chao Wei-Min.An Efficient Embedded Bit stream Parsing Processor For MPEG-4 Video Decoding System.International Symposium on VLSI Technology,System and Applications.2003

[11]ISO/IEC JTC 1/SC 29.14496-2-2004.Information technology-Coding of audio-visual objects-Part 2:Visual

[12]余兆明,李晓飞,陈来春.MPEG 标准及其应用.北京.北京邮电大学出版社.2002.50～128

[13]李宾,高平.H.264 编码系统的特点及其应用前景.电视技术.2003.6.19~21

[14]T.Wiegand,H.Schwarz,A.Joch,F.Kossentini,and G.J.Sullivan.Rate-constrained coder control and comparison of video coding standards.IEEE Transactions on Circuits and Systems for Video Technology.July 2003.vol.13, no.7.688～703

[15]严晓飞.H.264 码流结构的分析及其实现的优化[学位论文].西安.西安电子科技大学.2007

[16]I.E.G.Richardson.H.264 and MPEG-4 Video Compression-Video Coding for Next-generation Multimedia.Aberdeen,UK.The Robert Gordon University.2003.159～222

[17]柏海涛.H.264/AVC 编解码算法分析与优化[学位论文].四川.四川大学.2005

[18]闫冬.H.264/AVC 编码器的优化设计及实现[学位论文].成都.电子科技大学.2004

[19]余兆明.移动数字电视技术.北京.人民邮电出版社.2007

[20]毕厚杰.新一代视频压缩编码标准-264/AVC.北京.人民邮电出版社.2005

[21]Teng Chia-Yuan.An improved bloke Prediction mode for H.264/AVC Intra-frame Predietion.DCC.2004.3.569～569

[22]A.Luthra,G.J.Sullivan,T.Wiegand.Introduction to the special issue on the H.264/AVC video coding standard.IEEE Transactions on Circuits and Systems For Video Technology.2005

[23]A.Ahmad.Efficient Block Size Selection in H.264 Video Coding Standard.Electronics Letters.

Jan,2004.vol.40, no.1.19～21

[24]H.Malvar,A.Hallapuro,M.Karczewicz,and L.kerofsky.Low-Complexity Transform and Quantization in H.264/AVC.IEEE Transactions on Circuits and Systems for Video Technology.July 2003

[25]S.M.Henrique,H.Antti,K.Marta,and K.Louis.Low-Complexity Transform and Quantization in H.264.IEEE Trans.Circuits Syst.video Technol.July, 2003.vol.1,no.3.598～603

[26]傅祖芸.信息论—基础理论与应用.北京.电子工业出版社.2001.8

[27]刘峰.视频图像编码技术及国际标准.北京.北京邮电大学出版社.2005

[28]吴乐南.数据压缩.南京.东南大学出版社.2000

[29]JVT Reference Software JM 11.0.available online at: http://iphome.hhi.de/suehring/tml/download/

[30]D.Marpe,H.Schwarz,and T.Wiegand.Context-adaptive binary arithmetic coding in the H.264/AVC video compression standard.IEEE Trans.Circuits Syst.Video Technol.July,2003.vol.13.62～63

[31]丁贵广,计文平,郭宝龙.Visual C++6.0 数字图像编码.北京.机械工业出版社.2004.79～91

作者简历

钮任飞.男.生于1985年6月24日.汉族.

教育经历：

2007年9月—2009年7月	北京交通大学	通信与信息系统专业	硕士学位
2003年9月—2007年7月	河北大学	电子信息科学与技术专业	学士学位

基于H.264的熵编解码的研究及优化

作者： 钮任飞
学位授予单位： 北京交通大学