北京大学

硕士研究生学位论文

题目：H.264/AVC 至 H.265/HEVC 的快速转码算法研究

姓　　名：陈蕾

学　　号：1201213644

院　　系：深圳研究生院

专　　业：计算机应用技术

研究方向：多媒体信息处理技术

导师姓名：马思伟　教授

二〇一五 年 六 月

版权声明

摘要

近年来，随着软、硬件处理能力的增强，传统的标清视频已经不能满足人们日益增长的需求，高清视频甚至超高清视频将日益常见。然而，上一代的视频编码标准 H.264 的设计不足以满足高清和超高清视频的压缩编码，因此，新一代视频编码标准 HEVC 于 2013 年制定完成。在提供相同客观质量码流的同时，它比 H.264 可以节省近一半的码率。因此，在可以预见的未来，它必将取代 H.264 在工业界绽放异彩。然而，目前市场主流的编码器仍大部分采用 H.264/AVC 编码标准。H.264/AVC 至 H.265/HEVC 的快速转码算法对于分别使用 H.264 标准的应用设备和 HEVC 的应用设备之间的互操作性必不可少。此外，将 H.264 码流转码为 HEVC 码流还可有效减少存储成本及传输带宽，对工业界有着十分重要的意义。基于此，本文研究 H.264 至 HEVC 的快速转码技术，共提出三种快速转码解决方案。

首先，我们提出一种快速帧间转码技术：基于 AMV-RDO (Average Motion Vector Rate Distortion Optimization) 率失真模型的快速帧间转码算法。对于 H.264 至 HEVC 的转码应用，其最大难点在于由 HEVC 的高复杂度而带来的转码速率低的问题，该算法利用 H.264 的解码运动矢量信息来进行 PU 及 CU 的率失真建模，以取代 HEVC 的传统率失真模型。由于 AMV-RDO 率失真模型省去了复杂度极高的运动估计过程及变换、量化、解码过程，因此，它比 HEVC 的原率失真模型更快速。实验结果表明，该算法在引入一定转码质量损失的同时，转码时间节省为 87%，因此，十分适用于具有实时需求的快速转码应用。

其次，本文提出一种固定阈值下的快速转码算法，该算法基于统计特征进行帧内及帧间的快速转码。对于深度小于 2 的 CU，利用当前 CU 的特征与离线训练出的阈值之间的关系，进行 CU 的划分决策与 PU 的选择，对于深度大于或等于 2 的 CU，直接利用当前 CU 所对应的 H.264 宏块类型信息进行映射。实验结果表明，该算法在几乎不带来性能损失的同时，转码时间节省 58%。由于固定阈值下的快速转码算法在转码时间及转码性能之间取得了很好的平衡，因此，适用于对转码质量具有高要求的转码应用中。

再次，本文进一步提出一种阈值自适应的快速转码算法。固定阈值下的快速转码算法中，经过离线训练出的阈值将应用于所有序列的快速转码，固定

阈值是否适用于特征各不相同的输入码流仍是一个值得进一步探究的问题。为此，在本算法中，阈值将根据序列特征自适应调整，以获取更适合当前输入码流特性的阈值。实验结果表明，相比于固定阈值，本算法将带来更好的转码质量，但由于阈值需在线训练，因此，转码加速不如固定阈值转码。为进一步弥补阈值自适应转码速率低的问题，本文还提出一种 PU 快速模式决策算法以增强阈值自适应的快速转码的转码速率。

关键词：标准间转码，H.265/HEVC，H.264/AVC，快速转码，全解全编转码

Research on Fast Transcoding from H.264/AVC to H.265/HEVC

Lei Chen (Computer Applied Technology)
Directed by Siwei Ma

ABSTRACT

In recent years, with the enhancement of the software and hardware processing capacity, the SD(standard-definition) videos are unable to meet the growing needs of people, HD(high-definition) videos and even UHD(ultra high-definition) videos will become increasingly common. However, the design of H.264/AVC is not sufficient to meet the compression requirements of HD and UHD videos, therefore, the new generation of video coding standard HEVC is proposed and developed. Compared to H.264, HEVC can save nearly half of the bit-rate, while providing the same objective quality. We can see that HEVC has promising applications. Hence, not only to be ready to promote interoperability for the legacy video encoded in H.264/AVC format, but also to be able to take advantage of the superior rate-distortion performance of the HEVC, research on fast transcoding from H.264/AVC to H.265/HEVC is of great practical value. Here, we propose three transcoding schemes.

Firstly, a fast inter transcoding algorithm is proposed, namely fast inter transcoding based on AMV-RDO (average motion vector rate distortion optimization). For transcoding from H.264 to HEVC, the most difficulty lies in the high complexity of HEVC. In the proposed algorithm, motion vectors of decoded bit-stream are used to build the RD model of CU and PU in replace of the original RD model in HEVC. Since the time-consuming motion estimation process, transform and quantization process, entropy process, and decoding process are skipped, the transcoder is speed up. According to the experimental results, 87% transcoding time is saved with a little RD performance loss comprared with trivial transcoding.

Secondly, a fast transcoding algorithm based on fixed-threshold is proposed to

provide high quality transcoding. For CU in depth 0 and depth 1, the feature value of current CU is compared with the threshold to determine whether the CU should be further split or not and which PU mode should be selected. For CU in depth 2 and depth 3, a mode mapping table is designed and the CU and PU mode decision is directly determined by the corresponding MB type and subMB type. As the experimental results shows, compared with trivial transcoding, the transcoding time saving is 58% with nearly no RD performance loss.

Thirdly, a fast transcoding algorithm based on adaptive-threshold is proposed to further explore whether the off-line trained threshold can be applied to all input H.264 bit-stream. In the proposed algorithm, the threshold will be on-line trained, thus, the threshold is adaptively changed acooding to the different feature of different input bit-stream. According to the experimental results, the proposed algorithm achievers better RD performance compared with fixed-threshold transcoding. However, since the threshold is on-line trained, the transcoding speed is slower. To further accelerate the transcoding speed of adaptive-threshold transcoding, a fast PU mode decision algorithm is further proposed.

KEY WORDS: Heterogeneous video transcoding, H.264/AVC, H.265/HEVC, Fast transcoding, Trivial transcoding

目录

图目录

表目录

第 1 章 绪论

1.1 选题背景

视频是人们生活中密不可分的一部分,是人们进行休闲娱乐、获取信息的重要载体，与视频应用相关的技术往往能引起广泛的关注。随着计算机技术的快速发展，模拟视频已逐渐淡出历史舞台，取而代之的是数字视频。然而，数字视频的原始数据量之大是存储空间和传输带宽所难以承受的，如何有效进行数字视频的压缩，是数字视频领域的重要研究问题之一。

自上世纪 80 年代开始，经过国际电信联盟（International Telecom Union, ITU）视频编码专家组（Video Coding Expert Group，VCEG）和国际标准组织（International Standard Organization, ISO）运动图像专家组（Moving Picture Expert Group，MPEG）专家们的不懈努力，制定了一系列用于数字视频压缩的编码标准（如图 1.1 所示），并形成了以预测、变换、量化、熵编码为核心的混合编码框架。在 ITU 和 ISO 推出的一系列数字视频编码标准中，MPEG-2 标准[1]和 H.264/AVC 标准[2]均取得了工业界的广泛应用。由于 H.264/AVC 编码标准相比于 MPEG-2 标准在相同质量下提升了一倍的压缩效率，可以节省近一倍的存储空间和传输带宽，因而，它逐渐取代了 MPEG-2 标准，成为目前工业界的主流视频编码标准。

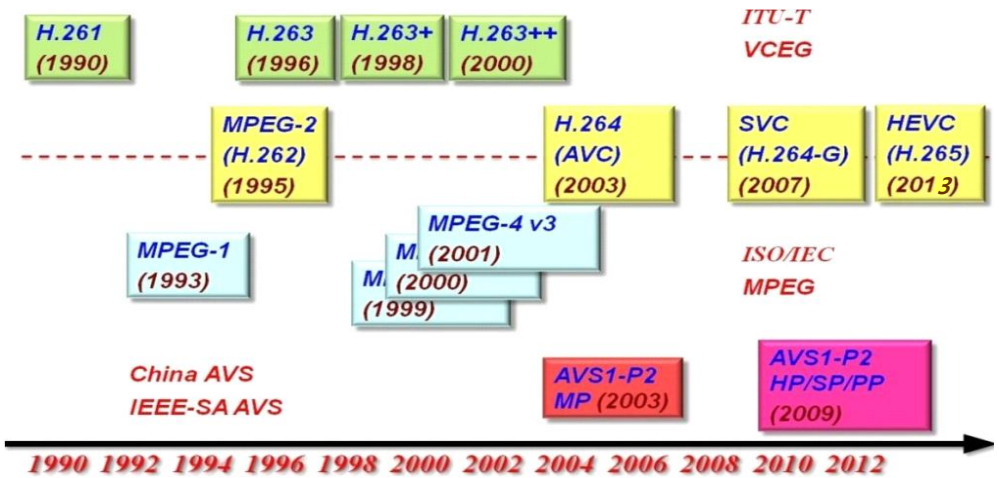


图 1.1 视频压缩标准的发展历程

近年来，随着软、硬件处理能力的增强，传统的标清视频（720x480,

720x576）已经不能满足人们日益增长的需求，高清视频（1280x720, 1920x1080）甚至超高清视频（3840x2160, 7680x4320）愈来愈常见。然而，MPEG-2，H.264/AVC 等视频编码标准的设计不足以满足高清和超高清视频的压缩编码，研究制定出更高压缩效率的视频编码标准成为了一个新挑战。因此，ITU VCEG 和 ISO/IEC MPEG 于 2010 年联合成立了视频编码联合协作团队（Joint Collaborative Team on Video Coding，JCT-VC），以致力于新一代视频编码标准——高效视频编码标准（High Efficiency Video Coding，HEVC）的制定。该标准旨在提供比 H.264/AVC 更高的压缩效率，即在提供相同质量的视频标准的前提下，提升一倍的压缩效率。2013 年 1 月，HEVC[3]（也称为 H.265）标准制定完成。由于 H.265/HEVC 编码标准较其它早期的视频编码标准均有更高的压缩效率[4]，在可以预见的将来，它必将取代 H.264/AVC 视频编码标准，在工业界获得广泛的应用。

然而，H.264/AVC 标准是一代非常成功的视频编码标准，在物理媒体、有线电视服务、网络流媒体中，均取得了非常广泛的应用。若能将由 H.264/AVC 压缩的视频转码成为符合 H.265/HEVC 标准的视频，那么，这些视频所消耗的存储成本及传输带宽成本均可大幅减少，因而，H.264/AVC 至 H.265/HEVC 的转码算法研究，对工业界有着十分重要的应用价值。

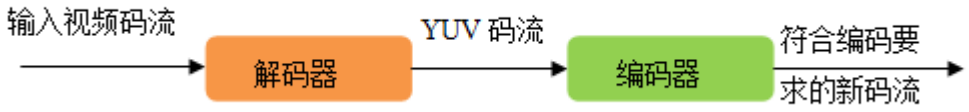


图 1.2 全解全编转码框架图

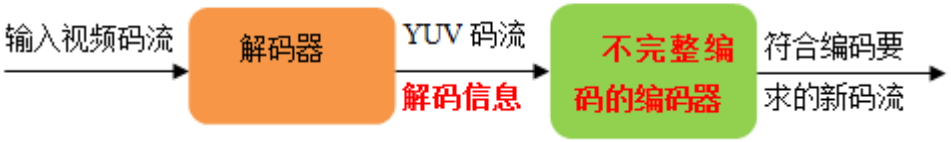


图 1.3 快速转码框架图

转码算法可以分成两大类，全解全编转码和快速转码。如图 1.2 所示，全解全编转码方式将输入的码流进行完全解码，再按照编码要求，将解码出来的 YUV 码流进行完全编码。因而，它是压缩性能最优的转码方式，但由于进

行了完整的解码过程和完整的编码过程，它也是速度最慢的转码方式。如图1.3所示，快速转码算法利用解码出来的信息来加速编码的过程，因而，它比全解全编的转码方式要快，但由于编码过程不完整，它的压缩性能略低于全解全编的方式。对于 H.265/HEVC 视频编码标准来说，尽管它拥有十分高效的压缩性能，它的编码复杂度也远高于早期的视频编码标准。因此，全解全编的转码方式显然不适用于 H.264/AVC 至 H.265/HEVC 的转码，如何能在编码复杂度和转码性能之间找到一个合适的平衡点，是本文研究的重点。

1.2 H.264/AVC 及 H.265/HEVC 编码标准简介

1.2.1 基于块的混合编码框架

H.264/AVC 标准[2-3]和 H.265/HEVC[4-5]标准均采用如图 1.4 所示的基于块的混合编码框架。在该框架中,主要包含预测、变换、量化和熵编码这四个核心模块。预测模块又分为帧内预测和帧间预测，帧内预测消除视频帧内的空间冗余，帧间预测消除视频帧与帧之间的时间冗余。变换模块通过减少像素间的相关性进一步消除视频帧内的空间冗余。量化模块减少需要编码的数据量从而达到数据压缩的目的。需要注意的是量化是一种有损压缩技术，会导致重构图像的失真，一般量化步长越长，重构图像的失真越大。最后，熵编码模块利用信源的信息熵进行码率压缩从而消除编码冗余。其具体流程为：

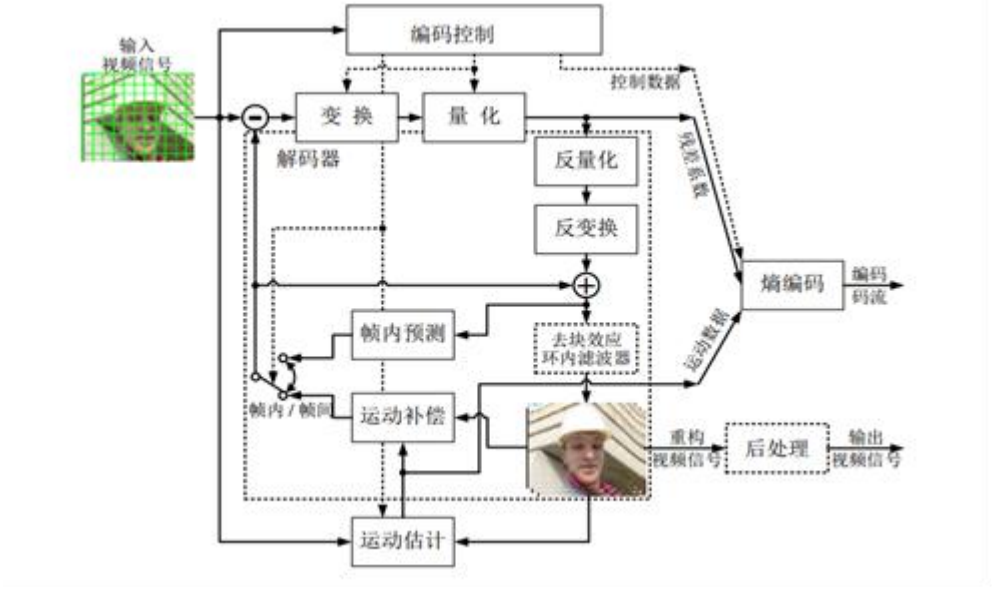


图 1.4 基于块的混合编码框架图

1. 由视频源输入一帧图像，每帧图像被划分成一个个块依次进入编码器进行编码。在 H.264/AVC 标准中，图像被划分成宏块（Macroblock, MB），每个宏块包含一个 16x16 像素的亮度块和 2 个 8x8 像素的色度块。在 H.265/HEVC 标准中，图像被划分成编码树单元（coding tree unit, CTU），其大小可以在编码配置文件中进行设定。

2. 通过编码控制模块来确定当前编码块是采用帧内编码还是帧间编码。如果采用帧内编码，则利用当前图像已经编码的部分来对当前编码块进行预测。如果采用帧间编码，则在缓冲区中找到和当前编码块最相似的重构块作为当前编码块的预测块，并同时获得运动矢量信息和参考帧信息。预测之后的预测残差块再进行变换、量化处理。

3. 将量化后的残差系数和预测信息（帧间预测的运动矢量信息、参考帧信息和帧内预测的方向信息等）送入熵编码器进行熵编码，形成最终的编码码流。

4. 在编码端中一般还包含了解码器，将量化后的系数进行反量化和反变换得到残差块，再与帧内或帧间预测获得的预测块相加得到重构块。将重构块存入缓冲区中，用以之后的编码块的预测。

5. 对于重构块，一般还需要进行环内滤波用以消除人工块效应。图 1.5(a)是未进行滤波后的重构帧，图 1.5(b)是进行滤波后的重构帧。对比两图可以发现，通过滤波可以有效消除块效应，提升主、客观图像质量。环内滤波后的重构图像用于之后编码块的预测。除了环内滤波，还可以在图像显示前进行滤波处理操作，我们称这种操作为后处理操作，它是一种环外滤波操作。

(a) **(b)**

图 1.5 不采用(a)和采用(b)去方块滤波器的 H.264 编解码器的效果

1.2.2 H.264/AVC

1.2.2.1 块大小

在 H.264/AVC 的混合编码框架中，宏块(Macroblock, MB)是进行编码的基本单位。它包含一个 16x16 像素的亮度块和两个 8x8 像素的色度块。对于一帧图像，它被划分成一个个宏块，依次进入编码框架中进行预测、变换、量化、熵编码等过程。

1.2.2.2 帧内预测

帧内亮度块共有三种编码类型：I_4x4、I_16x16 和 I_PCM。I_4x4 类型将一个宏块划分成 16 个 4x4 块，每个 4x4 块分别进行预测，这种模式适用于细节丰富的图像部分。I_16x16 类型对整个 16x16 亮度块进行预测，适用于图像平坦部分。I_PCM 类型不进行当前块的预测，而是直接将当前宏块进行变换、量化处理，适用于不规则图像的编码。对于 I_4x4 块，它共有 9 种帧内预测方向模式，而 I_16x16 块只有 4 种预测方向，如图 1.6 和 1.7 所示。对于每一个帧内预测宏块，它既可以选择 I_4x4 模式，也可以选择 I_16x16 模式，还可以选择 I_PCM 模式，最终所选模式由编码控制单元决定。在之后增加的 H.264 High Profile[2]中又增加了 I_8x8 类型，与 I_4x4 类型相同，每个 I_8x8 块也包含 9 种预测方向模式。

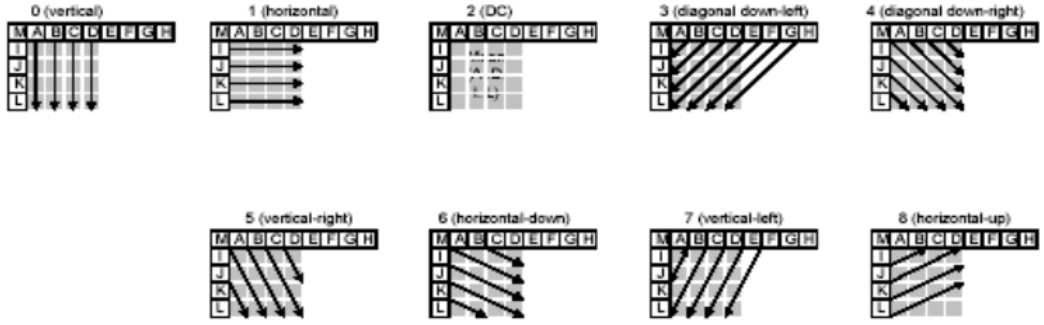


图 1.6 H.264/AVC 中 4x4 亮度块的帧内预测模式

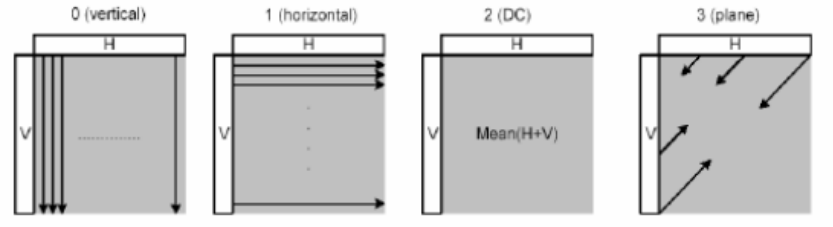


图 1.7 H.264/AVC 中 16x16 亮度块的帧内预测模式

帧内色度块的预测独立于亮度块预测，它基于 8x8 块进行。但与亮度 I_8x8 块不同，色度块只包含 4 种预测方向模式：水平模式、垂直模式、DC 模式和平面模式(plane mode)。其具体预测方法与帧内亮度块的 I_16x16 块的四种方向预测模式相同。

1.2.2.3 帧间预测

除了 1.2.2.2 中提到的 I 宏块类型，还有 P 宏块类型和 B 宏块类型。P 宏块类型和 B 宏块类型均进行帧间预测技术。对于 P、B 宏块来说，它们不再使用本帧内已经编码的周围像素点进行当前编码块的预测，而使用运动估计(motion estimation, ME)技术[6]和运动补偿(motion compensation, MC)[7-8]技术来获取预测块。在 H.264/AVC 中，亮度块运动补偿以 1/4 像素点为单位，色度块以 1/8 像素点为单位。由于运动补偿不再以整像素点为单位，因此，在进行运动估计之前必须进行非整像素点的插值。在 H.264/AVC 中，分像素点采用 6 拍滤波器插值获得，1/4 像素点利用邻近的整像素点和分像素点求平均获得。色度块的非整像素点的插值则采用双线性插值方法。

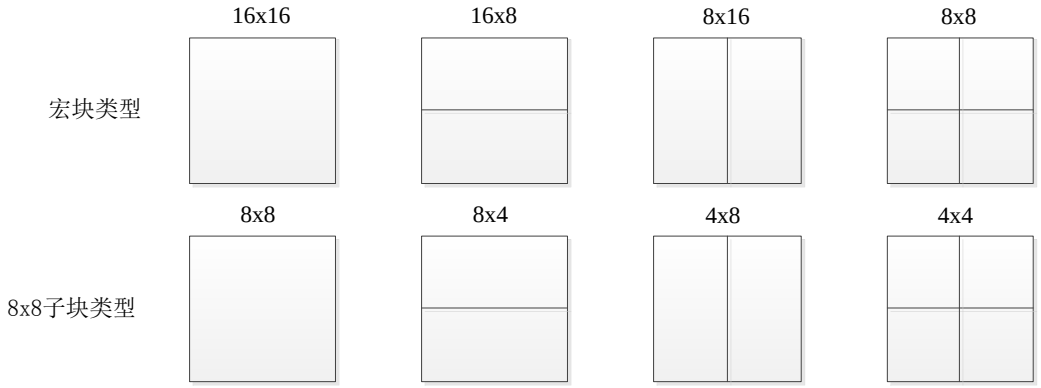


图 1.8 宏块的划分，上图为宏块划分成块，下图为 8x8 块划分子块

在 H.264/AVC 的帧间预测中，一个宏块被划分成块，每个块独立进行预测，即独立进行运动估计和运动补偿操作，拥有各自的运动矢量和参考帧索引信息。对于一个帧间宏块，它有多种划分方式：划分成一个 16x16 块、二个 16x8 块、二个 8x16 块，或者四个 8x8 块。对于每一个 8x8 块，它还可以进一步划分：划分成一个 8x8 子块、两个 8x4、两个 4x8 子块，或者四个 4x4 子块，如图 1.8 所示。假设一个宏块被划分成四个 8x8 块，每个 8x8 块又进一步划分成四个 4x4 子块，那么该宏块总共需要传输 16 组运动矢量值和参考帧索引值。

H.264/AVC 在传统的跳过模式(skip 模式)基础上，还引入了 B_direct 模式，该模式只可用于 B 宏块。另外，H.264/AVC 还引入了通过 B 帧技术[9]，即 B 帧可作为其它 B 帧的参考帧。此外，P 宏块的参考帧的数目从原来的一帧变成多帧，B 宏块则由前后的各一个参考帧变成前后各一队列的参考帧。这些技术，都为 H.264/AVC 提供更高精度的帧间预测作出了贡献。

1.2.2.4 变换与量化

DCT 变换是一种性能最接近于 K-L 变换的变换方式，因而，H.264/AVC 之前的标准均采用 8x8DCT 变换。但是，DCT 变换会引入无理数浮点除法操作，从而导致各解码器解码码流不一致。为解决这种归一化问题，H.264/AVC 采用 4x4 整型变换[10]，它只需整数的加法和移位操作，因而，可以避免由于解码处理器不同而导致的解码码流不一致的问题。对于色度残差块和采用 I_16x16 预测模式的亮度残差块，它除了需要进行 4x4DCT 变换外，还需要把各个 4x4 块的 DC 系数组合起来再进行一次哈德玛变换(Hadamard transform)，以进一步减小各 DC 系数间的空域相关性。我们将 I_16x16 和色度残差块的这种变换方式称为两次变换操作。由于色度块的大小为 8x8 像素，它只包含 2x2 个 DC 系数，因此，对于色度块的二次变换，H.264/AVC 还引入了一个 2x2 哈德玛变换矩阵。

此外，在 H.264/AVC 中还将变换与量化这两个独立的步骤合二为一，以进一步减少编解码的运算量。在之后增加的 H.264 High Profile 中，还引入了整数 8x8 变换。

1.2.2.5 熵编码及环内滤波

H.264/AVC 支持两种熵编码方式：基于上下文自适应的可变长熵编码

(context-adaptive variable length coding, CAVLC)和基于上下文自适应的算术熵编码(context-adaptive binary arithmetic coding, CABAC)。CAVCL 用于经过变换、量化后的系数的编码。不同于以往单一码表的熵编码方式， CAVLC 根据已经编码的句法元素进行码表的切换以实现其自适应的方式，进一步提高熵编码的编码效率。CABAC 可以为字母表中的每个符号分配非整数长度的比特，因而，更适用于概率高于 0.5 的符号的熵编码表示。CABAC 在编解码端用相同的方法估计和更新概率表，以实现概率模型的自适应。更多关于 CAVLC 和 CABAC 的细节可以查看参考文献[11]。

H.264/AVC 的环内滤波采用去块滤波技术(deblocking filter)。该技术以宏块为单位，在每个 4x4 块边界进行水平方向和垂直方向的滤波，以消除基于块的编码方式所引入的块效应，增强视频的主观质量。其具体技术细节可参考文献[12]。

1.2.3 H.265/HEVC

1.2.3.1 块大小及划分

与 H.264/AVC 采用宏块作为编码单位不同，H.265/HEVC 的编码单位为编码树单元(coding tree unit, CTU)[13]。一个 CTU 包含一个亮度编码树块(coding tree block)和两个色度编码树块。亮度 CTB 的大小由编码端的配置文件决定，可以为：16x16 像素、32x32 像素、64x64 像素。色度 CTB 的大小与采样格式有关，在 4:2:0 的采样格式中，其长宽各为亮度 CTB 的一半。对于一个 CTU，它通过一种四叉树递归划分结构进一步划分成一个或多个编码单元(coding unit, CU)，每个编码单元包含一个亮度编码块(coding block, CB)和两个色度编码块。每个编码单元作为四叉树的一个节点，还可进一步递归划分成四个更小的 CU。当亮度编码块达到编码器允许的最小值时，CU 节点不再往下划分，递归过程结束。由于每一个四叉树 CU 节点均可选择是否进一步划分，因而，一个 CTU 的四叉树划分结构可能有许多种，最终写入码流的递归划分结构由编码控制单元决定。亮度编码块的最小值由编码端配置文件决定，通常是 8x8 像素。

图 1.9 是一个 CTU 的四叉树递归划分示例。左图是 CTU 的最终递归划分结果，右图为对应的四叉树结构。从图中可以看到，最高层的 CU 结点进一步划分成四个深度为 1 的 CU。最左边的深度为 1 的 CU 选择进一步划分成四

个深度为 2 的 CU，而第 2 个深度为 1 的 CU 最选择不继续划分，第 3、4 个深度为 1 的 CU 均选择继续划分。深度为 2 的第 1 个 CU 也选择继续划分成 4 个深度为 3 的 CU，而第 2 个 CU 则终止划分，其它深度为 2 的 CU 的划分结果可以根据左图以此类推。对于深度为 3 的 CU 均选择不继续划分，因为它的亮度编码块已经达到了最小值，必须终止递归过程。

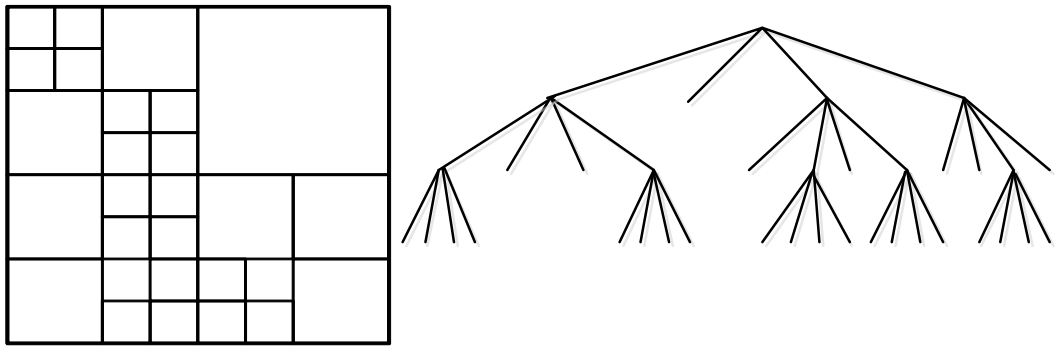


图 1.9 HEVC 的 CTU 四叉树划分示例

1.2.3.2 帧内预测

H.265/HEVC 中进行预测的基本单位为预测单元(prediction unit, PU)，它包含一个亮度预测块(prediction block, PB)，两个色度预测块和对应的句法元素。对于帧内预测，它只有两种 PU 模式：PART_2Nx2N 模式和 PART_NxN 模式。当 PU 模式为 PART_2Nx2N 时，亮度预测块大小等于它所在 CU 的亮度编码块大小。当 PU 模式为 PART_NxN 时，当前 CU 进一步划分成四个 PU，每个 PU 独立进行帧内预测，其大小等于它所在 CU 长宽的各一半。通常情况下，帧内预测的 PU 模式为 PART_2Nx2N 模式，只有当亮度编码块大小为码流所允许的最小值时，PART_NxN 模式才可被选择。在这种情况下，码流中需要传输一个标记位，以表示当前 CU 是否进一步划分成四个 PU。

在 H.265/HEVC 中，亮度编码块的帧内预测的预测模式进一步扩展至 35 种[14]，如图 1.10 所示，包含 Planar 模式、DC 模式和 33 种方向模式。色度编码块共有 5 种预测模式，即 DC 模式、Planar 模式、垂直预测模式、水平预测模式、与亮度相同模式的预测模式。

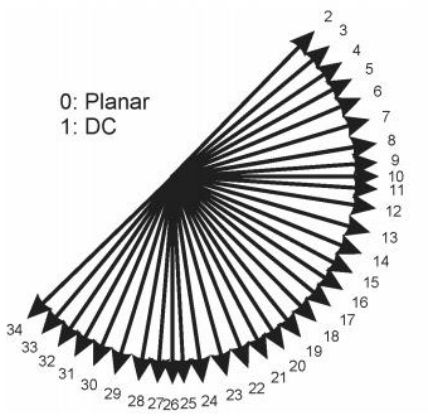


图 1.10 HEVC 中帧内预测的 35 种预测模式

1.2.3.3 帧间预测

H.265/HEVC 的帧间预测共支持 8 种预测模式：PART_2Nx2N, PART_2NxN, PART_Nx2N, PART_2NxnU, PART_2NxnD, PART_nLx2N, PART_nRx2N, PART_NxN，如图 1.11 所示。在 PART_2NxN, PART_Nx2N 模式下，CB 在水平或垂直方向上被划分成大小相同的两个 PB 块。在 PART_2NxnU, PART_2NxnD, PART_nLx2N, PART_nRx2N 模式下，CB 块被划分成大小不同的两个 PB 块，这种划分模式称为非对称运动划分模式 (asymmetric motion partitions, AMP)，它是 H.265/HEVC 新引入的帧间预测模式。帧间的 PART_NxN 模式的使用条件与帧内 PART_NxN 模式的使用条件相同。

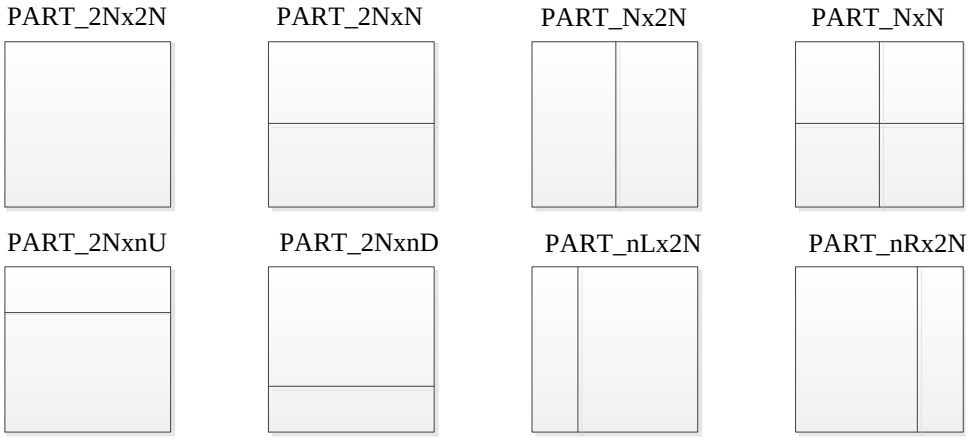


图 1.11 HEVC 中帧间预测的 8 种划分模式

与 H.264/AVC 相同，H.265/HEVC 中的亮度块也支持 1/4 像素精度的运动

估计和运动补偿技术。在采样格式为 4:2:0 的视频编码中，色度支持 1/8 像素精度的运动补偿技术。但不同于 H.264/AVC 的 6 拍滤波插值技术，H.265/HEVC 中采用 8 拍滤波器进行半像素点的插值，采用 7 拍滤波器进行 1/4 像素点的插值。对于色度分像素点，H.265/HEVC 采用 4 拍滤波器插值获得，而非 H.264 中的双线性滤波器。

此外，H.265/HEVC 的帧间预测还采用了两种新技术：Merge 模式[15]和 AMVP(advanced motion vector prediction)技术[16]。Merge 模式类似于 H.264/AVC 中的 skip 模式或 direct 模式，它利用空域相邻块和时域相邻块组成自己的候选集，在传输中，Merge 模式只需传输候选集的索引值即可。AMVP 技术与 Merge 模式类似，在进行运动矢量预测时，也可以利用空域相邻块和时域相邻块组成自己的运动矢量预测候选集。在不使用 Merge 模式的 CU 中，其运动矢量的预测值从多个预测候选中选择一个。

1.2.3.4 变换与量化

在 H.265/HEVC 中，进行变换的基本单位为变换单元(transform unit, TU)，它包含一个亮度变换块(transform block, TB)和两个色度变换块。与 CTU 的四叉树递归划分结构类似，TU 也通过四叉树递归划分来进行预测残差块的变换递归决策[17]。一个 CU 的预测残差块可以不划分直接进行变换、量化操作，也可以进一步划分成四个 TU，每一个 TU 还可选择是否进一步的递归划分。图 1.12 给出了一个 CTU 的四叉树递归划分示例及每个 CU 的 TU 递归划分示例。图中实线代表了 CU 的递归划分，虚线代表了 TU 的递归划分。变换块划分的最大深度需要在编码配置文件中进行设置，其大小包括：4x4 像素、8x8 像素、16x16 像素、32x32 像素。与 H.264/AVC 不同的是，H.265/HEVC 允许一个 TB 包含多个帧间 PB 块，以获得更高的编码效率。但对于帧内预测残差，TB 大小不能超过所在 CB 的 PB 大小。一般地，H.265/HEVC 中的变换核采用 DCT 整型变换，但对于帧内预测残差，还可以使用 4x4 DST 整型变换。由于 H.265/HEVC 采用变换四叉树递归划分结构，允许大块变换结构，还引入了 4x4DST 整型变换结构，因此，其变换编码效率进一步得到提升。

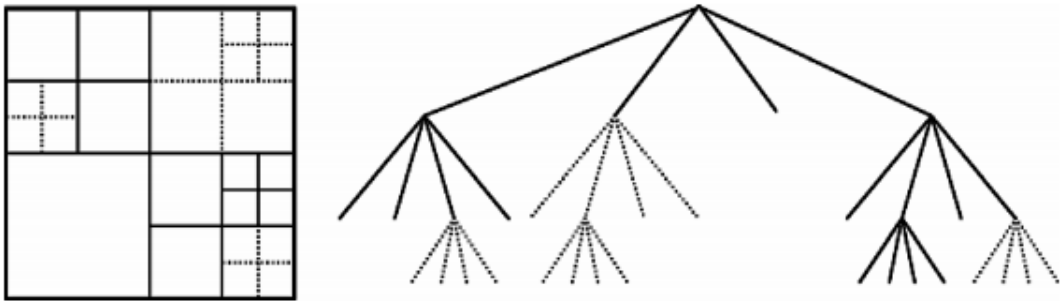


图 1.12 CTU 四叉树递归划分示例及每个 CU 的 TU 递归划分示例

1.2.3.5 熵编码与环路滤波

与 H.264/AVC 同时使用 CABAC 和 CAVLC 熵编码不同，H.265/HEVC 只使用一种熵编码方式：CABAC。其核心熵编码算法并未改变，具体技术细节可参考文献[18]。

H.265/HEVC 在环内滤波中新增加了样本自适应偏移滤波技术(sample adaptive offset, SAO)[19]，用于去块滤波器之后。去块滤波器一般只用于块边界处，以消除基于块的混合编码器所产生的边界块效应。但 SAO 滤波器自适应的对所有解码后的像素点增加一个偏移值，以达到图像显示增强的目的。H.265/HEVC 中去块滤波器及 SAO 技术级联使用，可以有效地增加图像的主观质量，这也是 H.265/HEVC 较 H.264/AVC 主观质量有很大提升的一个主要因素。

1.2.3.6 H.265/HEVC 性能及复杂度分析

CTU 的四叉树递归划分技术、帧内多方向预测技术、多模式的帧间预测技术、Merge 模式技术、AMVP 技术、TU 的四叉树递归划分技术、DST 变换技术及 SAO 滤波技术等均为 H.265/HEVC 引入的新技术。这些技术的引用带来了 H.265/HEVC 的高效压缩效率。据文献[20]表明，在主观质量评价下，H.265/HEVC 较 H.264/AVC 有 53%的码率节省，对于高分辨率的序列，主观码率节省甚至达到 67%。在客观质量评价下，AI、LD 和 RA 配置分别有 21.9%，34%和 37.1%的性能增益。

然而，这些新技术在带来高效压缩效率的同时，也引入了高复杂度。对于 CTU 四叉树中的每个 CU 节点在进行是否递归划分的决策时，需要进行当前 CU 的完整编码过程及四个子 CU 的完整编码过程，并分别计算当前 CU 的

RD 值与四个子 CU 的 RD 值之和，以判断当前 CU 节点是否需要进一步划分。对于四个子 CU 的递归划分过程也采用相同的方式，当所有 CU 的递归划分决策完成之后，才可以得到当前 CTU 的最优四叉编码树结构。对于每一个 CU 的完整编码，还需要进行帧内与帧间模式的决策。对于帧内预测，需要进行模式与预测方向的决策；对于帧间预测，需要进行多帧间模式的决策，及对应的运动估计和运补偿过程。这些技术都极大的增加了 H.265/HEVC 的编码复杂度，使其复杂度较 H.264/AVC 提高了诸多倍。倘若我们能利用已有的信息来进行四叉树递归划分决策的裁剪以及帧内、帧间模式的预测，减少 CU 节点递归划分决策的次数和模式决策的次数，那么，H.265/HEVC 的复杂度便可有效地降低。

1.3 转码技术

多媒体系统可以应用在工业界的各种领域，如：医疗领域、教育领域、交通领域等，对人们的学习、生活都有着深远的影响。它的一个基本职责是在任何时间、任何地点都能提供平稳、不间断的音视频流。然而，多媒体系统可由各式各样的设备组成，如：PC, 笔记本，智能手机，移动电脑等，这些设备之间通过各种有线或无线网络进行连接。在这样的多媒体系统中，码流在源端被压缩成某种指定的格式，但由于系统终端接入设备具有不同的显示能力、内存容量、处理和解码能力等，因而，必须对原始码流进行调整，以使其能在各终端设备上进行显示播放。这便是转码机制，它通过改变源码流的码率、帧率、分辨率甚至编码格式等，以使其适应不同的网络带宽和终端设备。转码技术并不是编码标准的一部分，但它对于多媒体系统的应用，却有着十分重大的意义。

图 1.13 视频转码框架图

目前，各种视频压缩标准并存于不同的多媒体应用设备上，随着多媒体应用数目的增加及有线、无线网络之间的相互融合，不同系统及平台之间的互操作性成为了一项新需求。转码技术通过调整已经编码视频的各项参数，

如图 1.13 所示，获得符合要求的码流，以满足各平台之间的互操作性。参考文献[20]指出，H.265/HEVC 较 H.264/AVC 在提供相同客观质量码流时，可以节省近 50%的码率。由于 H.265/HEVC 的高效压缩效率，在可以预见的未来，它必将拥有广泛的应用场景，在工业界绽放异彩。然而，目前工业界的主流编码器仍大部分采用 H.264/AVC 编码标准。H.264/AVC 至 H.265/HEVC 的快速转码算法对于分别使用 H.264/AVC 标准的应用设备和 H.265/HEVC 的应用设备之间的互操作必不可少。此外，将 H.264/AVC 码流转码成为 H.265/HEVC 码流还可有效减少存储成本及传输带宽，对工业界的成本节约也具有重要的意义。因而，研究 H.264/AVC 至 H.265/HEVC 的快速转码算法，具有重要的科研意义和应用价值。

1.4 研究内容

就标准间的快速转码算法而言，它一般需要满足三个要求：（1）尽可能的利用原始输入码流中的信息用于二次编码过程，以降低二次编码的复杂度，减少整体转码时间；（2）目标码流的质量应该尽可能的接近全解全编转码的码流质量，即在尽可能使用输入码流信息进行二次编码加速时，仍然需要保证转码后的视频码流质量；（3）对于有实时要求的应用，转码延迟和内存消耗应尽可能小。

1.2 节中曾详细介绍过 H.264/AVC 和 H.265/HEVC 标准中重要模块的具体技术，因而，我们可以总结到，H.264/AVC 标准和 H.265/HEVC 标准有许多相似点：均使用基于块的混合编码框架、帧内均使用多方向预测、帧间具有多种划分模式等。甚至，我们可以认为 H.264/AVC 标准是 H.265/HEVC 标准的子集：H.265/HEVC 的帧内预测的 35 种方向是在 H.264/AVC 的 9 种预测方向上进行的扩展；H.265/HEVC 的帧间预测的 PU 模式中的 PART_2NxN 模式、PART_Nx2N 模式在深度为 2 时，可以认为是 H.264/AVC 中的 16x8 和 8x16 块等等。因此，在 H.264/AVC 到 H.265/HEVC 的快速转码中，解码出的 H.264/AVC 码流中的各类信息，包括：预测信息、模式信息、残差信息等，均可用于 H.265/HEVC 编码的模式预测、四叉树决策中，从而达到降低 H.265/HEVC 编码复杂度的目的。

综上，本文研究 H.264/AVC 至 H.265/HEVC 的快速转码算法，核心思想是利用 H.264/AVC 的解码信息作用于 H.265/HEVC 的编码部分，从而减少编码复杂度，加速转码过程。具体分为两部分研究：（1）预测转码算法，以尽

可能快的速度进行标准间的转码。该算法利用解码信息进行编码端的 CTU 四叉树划分的预测，即预测四叉树的决策结果，最后，将预测的结果用于编码四叉树的裁减，从而实现转码加速。（2）基于阈值的转码算法，实现转码速度与质量的平衡。第一部分的转码算法通过预测四叉树的决策结果以实现快速转码，但若该方法一旦预测不准确，转码质量损失则会较大。这一部分的快速转码，通过学习、训练过程，得到一对较优的阈值，利用阈值进行编码四叉树的裁剪，即只裁剪掉一些不可能编码的部分，而将可能编码部分均进行尝试，从而，达到编码质量更优的转码器。在本部分中，阈值的确定，对于转码算法的性能尤为重要。因此，我们将本部分研究内容再细分为两个子类：（i）固定阈值的转码算法，该算法通过统计大量的序列，进行阈值的离线学习；（ii）自适应阈值的转码算法，该算法进行在线的阈值训练，因而，对不同序列的特征可以自适应的选择，从而获得更优的转码性能。

1.5 章节安排

第 2 章研究并分析经典的快速转码算法，包括标准内的快速转码算法及标准间的快速转码算法。第 3 章描述基于预测的快速转码算法，包括算法模型、算法具体实现及对应的实验结果。第 4 章描述基于阈值的快速转码算法，着重介绍阈值的学习、训练方式，以及基于阈值的快速转码算法的设计。第 5 章在上一章的基础上，进一步研究自适应阈值的快速转码算法。着重介绍自适应的方法及实现，并进一步给出实验结果及分析。最后，第 6 章对全文进行总结与展望。

第 2 章 转码技术研究现状

2.1 引言

在转码研究的初始阶段，一般通过将解码器与编码器直接串联的方式来实现转码技术，即通常所称的全解全编转码方式。在全解全编转码中，解码器将原始输入码流进行完整解码以得到解码码流，再根据转码要求对解码码流进行完全编码。这种转码方式的优点是转码后的视频质量最优，并且结构简单、利于实现，但由于完全没有利用原始码流信息，因此，转码复杂度最高，不利于实时转码的实现。与全解全编转码方式对应的是快速转码，它克服了前者复杂度高的缺点，通过将原始输入码流信息应用于编码端的方式进行转码加速。就应用而言，快速转码可以分成两类：标准内的转码及标准间的转码[21-23]。标准内的转码主要包括：降码率的转码、降帧率的转码和降分辨率的转码。这三种转码方法可根据实际应用需求，将原始输入码流转码成满足不同网络带宽、显示要求的码流。标准间的转码随着视频标准技术的发展应运而生，它将符合某一标准的视频码流转码成符合另一编码标准的视频码流，以充分利用各标准的压缩性能并实现多媒体设备之间的交互操作。

本章就标准内及标准间的转码技术的研究现状进行阐述及分析。通常情况下，标准间的转码技术也包含降码率、降分辨率和降帧率的转码。但是，由于各个标准之间的压缩技术各不相同，因此，本章在讨论标准间的转码时主要关注如何利用标准间技术的异同进行快速转码，而不再讨论标准间的降码率、降分辨率和降帧率的研究。

2.2 标准内的转码技术

2.2.1 降码率的转码技术

降码率的转码是一项应用非常广泛的转码技术。原始拍摄的视频往往具有很高的码率以供后期剪辑制作，但制作好的视频节目通常需要转换成不同码率的码流以供拥有不同网络带宽的用户观看。因此，降码率的转码技术研究如何有效地将高码率的码流转码成低码率的码流。就降码率的转码技术而言，一般包括三类转码框架：开环结构转码 [24-26]、像素域的级联转码 [25][27]

及 DCT 域的转码 [28-30]。

开环结构的转码框架图如图 2.1 所示，它不进行完整的解码过程与编码过程。在解码部分只进行熵解码和反量化操作，在编码部分只对反量化之后的离线余弦变换(Discrete Cosine Transform, DCT)系数进行量化及熵编码操作。在输出码流中，输入码流中的帧内或帧间预测信息被完全重用。此外，开环结构转码器还可对解码出的高频 DCT 系数直接丢弃，以进一步降低码率。由于不进行完整的解码过程和编码过程，并在编码时省去了复杂度较高的帧内预测和帧间预测，因此，开环结构转码器具有高效的转码速度。然而，编码时直接在频域上进行的量化操作修改了原始输入码流的预测残差，导致输出码流在解码后无法得到与编码端相同的重构图像，从而引发之后解码的帧间块无法获取最优预测块并导致重构块的质量进一步下降，并继续影响再之后解码的帧间图像质量。这种现象被称为漂移误差，它只会出现在帧间图像中，并将随着帧间图像的不断解码累积，导致转码后的码流质量不断变差。然而，这种漂移误差可以通过 I 帧进行消除，所以，开环结构的转码器适用于于 GOP 结构较短的视频码流，以及要求高速度、低内存的转码应用中。

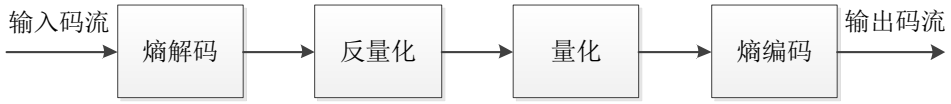


图 2.1 降码率转码中的开环结构转码框架图

为了消除开环结构转码中由于漂移误差导致的视频转码质量较差的问题，文献[25]提出了一种像素域的级联转码器。其转码框架图如图 2.2 所示。与开环结构转码器不同的是，它的解码部分需要进行完整的解码以得到重构 YUV 文件。在编码部分，除耗时较多的运动估计过程可以利用解码出的运动矢量进行代替，其它过程都需完整进行。由于在像素域转码中不存在编码预测残差与解码预测残差不一致的情况，因此，不存在漂移误差问题。像素域的级联转码器与全解全编转码器的差别在于，由于利用于解码的运动矢量信息省去了耗时最多的运动估计操作，因而，在消除漂移误差保证图像质量的同时，转码速度得到提升。此外，为进一步提升转码后的视频质量，文献[31]还提出一种运动矢量修正的方法，它利用解码后的运动矢量在小范围内进行运动估计，以弥补直接使用解码运动矢量带来预测块不准确的问题。

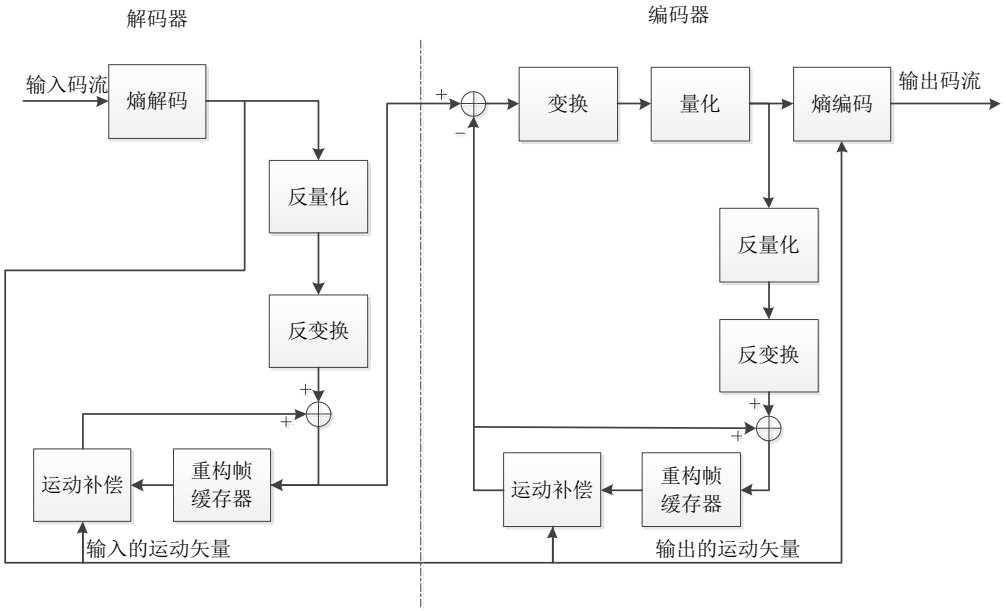


图 2.2 降码率转码中的像素域的级联转码框架图

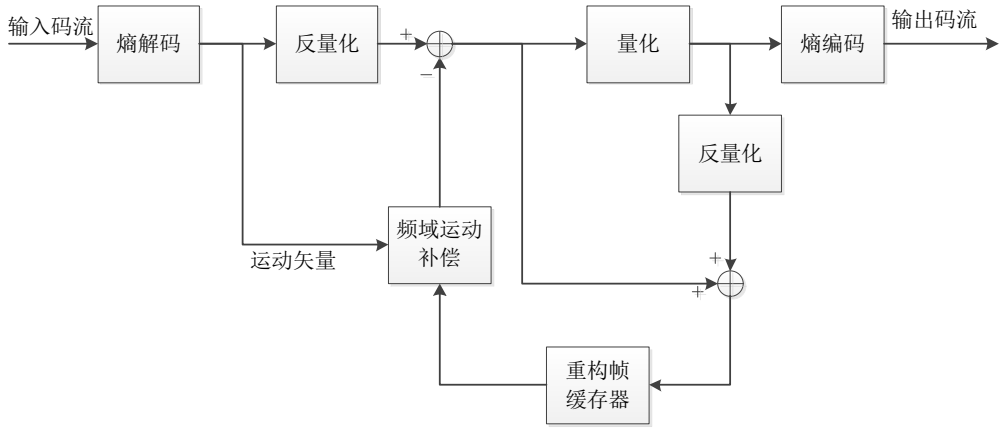


图 2.3 DCT 域的降码率转码框架图

DCT 域转码基于变换、反变换和运动补偿模块均是线性操作的假设进行。它最初由文献[28-29]提出，并在文献[30]中得到简化，其具体框架图如图 2.3 所示。在 DCT 域转码中，变换和反变换操作均不进行，且与像素域的级联转码相比只需一半的缓存空间。从框架图中可以看到，DCT 域的运动补偿操作是转码部分最耗时的模块，为此，文献[32-36]提出了快速算法以进一步加速该模块。鉴于 DCT 域转码可使用多种快速 DCT-MC 算法进行速度优化，因此，它比像素域级联转码器更快，且需要的内存空间也更小。然而，变换、反变换及运动补偿操作的线性假设并不完全正确，因此，DCT 域的转码器会引入漂移误差，转码后的图像质量略低于像素域的级联转码器。

2.2.2 降帧率的转码技术

高清摄像机采集的视频帧率通常为 60 帧每秒(frame per second, f/s)，但各种显示设备的视频帧率播放能力通常不同，例如，电脑上播放的电影视频帧率为 24f/s，手机上视频的帧率一般为 15f/s，而电视节目的帧率为 25f/s，因而，原始采集的高帧率视频需要转码成低帧率的视频以便在各种显示设备上进行播放，降帧率的转码技术研究的便是这一课题。在降帧率的转码中，由于帧率的降低，输入码流中的一些帧必须丢弃以得到输出码流，这就导致指向这些被丢弃帧的运动矢量无效，如何利用指向丢弃帧的运动矢量导出指向有效帧的运动矢量是降帧率转码的核心问题，图 2.4 简明表述了此问题。目前，该问题的解决方法主要分成四类：双线性插值法 [37]、前向优先矢量选择(Forward Dominant Vector Selection, FDVS)法[38]、可伸缩的矢量组合(Telescopic Vector Composition, TVC)法[39]及活动性优先的矢量选择(Activity-Dominant Vector Selection, ADVS)法[40]。这四种方法均利用被无效运动矢量指向的丢弃帧中的预测块 S 所覆盖的四个块 S1-S4 的运动矢量导出指向有效帧的运动矢量。

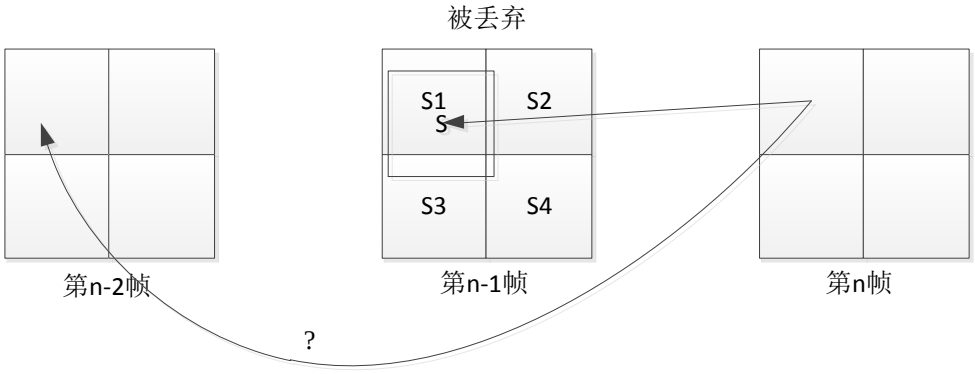


图 2.4 降帧率转码时运动矢量推导问题示例

双线性插值技术利用预测块 S 与被覆盖的四个块 S1-S4 的相对位置，将四个块 S1-S4 的运动矢量插值获得指向未丢弃帧的运动矢量。此外，为提高运动矢量的精度，该方法还进行一步利用导出的运动矢量作为搜索中心点进行小范围的运动估计过程。

图 2.5 是 FDVS 的示意图，n-1 帧和 n-2 帧均为被丢弃帧，n-3 帧为有效帧。因此，需要导出当前编码块指向第 n-3 帧的运动矢量。其具体执行方法为：比较预测块 S 与四个被覆盖块 S1-S4 的重叠度，重叠度最高的块的运动矢量作为当前编码块指向 n-2 帧的运动矢量，此示例中 MV2 是推导出的当前

编码块指向第 n-2 帧的运动矢量。若 n-2 帧仍是被丢弃帧，再重复该过程，直到推导出的运动矢量指向有效帧。

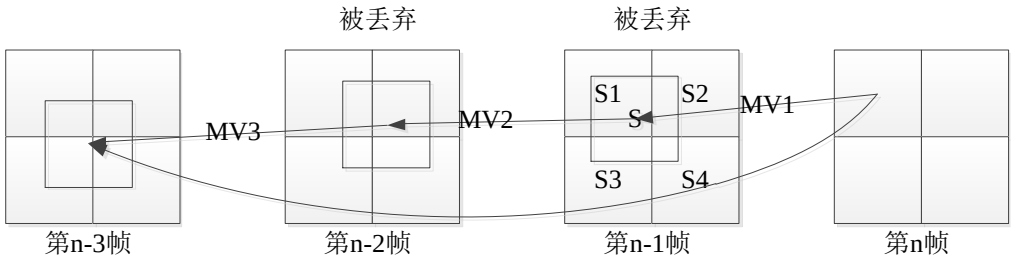


图 2.5 前向优先矢量选择方法示例

TVC 将丢弃帧中的预测块 S 所覆盖的四个块 S1-S4 的运动矢量直接进行矢量相加，得到指向第 n-2 帧的运动矢量。若第 n-2 帧仍为无效帧，则重复该过程，直到导出的运动矢量指向有效帧。

ADVS 根据预测块 S 所覆盖的四个块 S1-S4 的活动性来推导运动矢量。块的活动性一般利用该块所包含的非零系数数目来判别，通常所含非零系数数目越大，表明其活动性越高。在 ADVS 中，S1-S4 中活动性最高的块的运动矢量作为当前编码块指向第 n-2 帧的运动矢量，若第 n-2 帧仍无效，重复此过程。

对比这四种运动矢量导出方法[23]，双线性插值法需要存储被丢弃帧中的所有运动矢量信息，内存消耗较大。FDVS 的性能优于双线性插值法，并且消耗更少的内存。TVC 比 FDVS 消耗更少的内存，但性能略低于 FDVS。ADVS 在大多情况下推导出的运动矢量与 FDVS 相同，但在图像活动较高的情况下，性能略优于 FDVS。

2.2.3 降分辨率的转码技术

降分辨率的转码将原始拍摄的高分辨率视频转码成低分辨率视频以便在各种显示设备上进行播放。由于分辨率的降低，输入码流的多个块合成一个块，因此，在转码过程中会带来运动矢量组合及修订问题、模式决策问题等。图2.6表明在2：1降分辨率时，四个运动矢量组合成一个新运动矢量的问题。该问题一般有5种处理方法：随机法[41]、中值法[39][42-43]、加权平均法[42,44]、加权中值法[39,41,45]及DC最大法[40]。利用这些方法推导出的新运动矢量可作为参考帧的初始搜索点，通过在初始点附近进行小范围运动估计在保证转码速度的同时可得到更优的运动矢量[31]。此外，上述五种运动矢量推导方法均基于2：

1的降分辨率转码，文献[46]还提出了一种任意比例降分辨率转码中运动矢量的组合及修订方法。

除了运动矢量的组合及修订问题，在2：1降分辨率转码中，由于四个块的模式可能各不相同，如图2.7所示，降分辨率后新块的模式如何决策也是一个重要研究点，文献[47]提出一种帧内模式决策的解决方法，文献[48][49]提出一种帧间模式决策的方法。

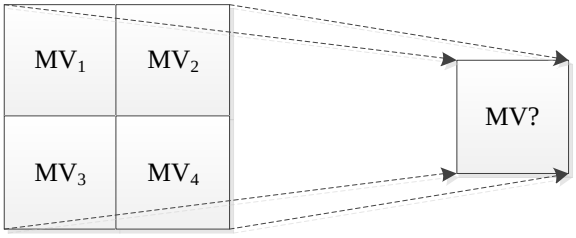


图 2.6 2：1 降分辨率转码中四个运动矢量合成一个新运动矢量示意图

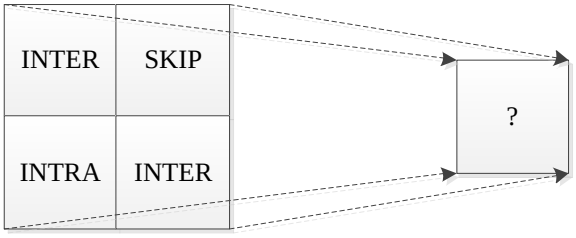


图 2.7 2：1 降分辨率转码中四个块模式合成一个新模式示意图

2.3 标准间的转码技术

在 20 世纪 90 年代，互联网及数字电视均未全面普及，人们一般通过 DVD 进行数字视频的观看， MPEG-2 编码标准压制的 DVD 视频由于具有较高的码率而不利于网络传输，为此，较 MPEG-2 压缩效率提升一倍的 H.264/AVC 视频编码标准得以制定并广泛使用，由此，数字视频时代正式被开启。与 H.264 相对应的是我国自主研制的 AVS 视频编码标准，它与 H.264/AVC 的压缩效率相当，由于是我国自主研制的，因此，它在应用中可以节省大量的专利费用。H.264/AVC 和 AVS 编码标准可以满足一般标清视频的压缩需求，然而，高清和超高清视频的码率仍非常大，不利于传输与存储，为此，HEVC 编码标准应运而生。由于在视频编码标准的发展过程中争先涌现了各种视频编码标准，当前存在一种或多种主流编码标准，且主流编码标准与新编码标准共存的现

象十分普遍，如何进行各标准之间的相互转码，已日渐成为视频转码领域的新研究热点。本节主要就广泛使用的四种视频编码标准间的快速转码技术进行介绍与分析，分为：MPEG-2 至 H.264/AVC 的快速转码技术、H.264/AVC 与 AVS 的相互转码技术及 H.264/AVC 至 HEVC 的快速转码技术。

2.3.1 MPEG-2 至 H.264/AVC 的快速转码技术

MPEG-2 标准主要应用于 DVD 中，由于其压缩效率较低不利于网络传输，伴随着网络技术的迅速发展 H.264/AVC 编码标准于 2003 年被制定完成。在相同视频质量下，H.264 比 MPEG-2 节省一倍码率，因而，H.264/AVC 视频码流可以满足网络传输需求。为将传统 DVD 视频转码成为 H.264 视频，MPEG-2 至 H.264/AVC 的快速转码技术在 21 世纪初曾被广泛研究，其转码技术主要分成帧内快速转码及帧间快速转码两类。

帧内快速转码指 MPEG-2 中的 I 帧至 H.264 中 I 帧的快速转码技术，它包括两部分：帧内快速模式决策算法与帧内预测方向快速决策算法。帧内快速模式决策算法决策当前帧内宏块的预测块大小，即编码成一个 16x16 预测块还是 4 个 8x8 预测块还是 16 个 4x4 预测块。帧内预测方向快速决策算法是指快速决策当前帧内预测块使用的预测方向。文献[50]提出一种利用 H.264 中当前编码宏块所对应的 MPEG-2 解码 8x8DCT 块的 DC 系数的方差与阈值之间的关系来决策帧内模式的算法，并进一步使用对应的 MPEG-2 解码 8x8DCT 系数来计算对应大小块的角度特征，从而决策当前预测块的预测方向。由于 8x8 大小的 DCT 系数只能计算 8x8 块大小的角度特征，对于 4x4 块和 16x16 块大小的角度特征无法利用此信息获得，因而，此文还提出一种利用 8x8DCT 块划分 4x4DCT 块及重组 16x16DCT 块的方法，以得到对应 4x4 预测块和 16x16 预测块的角度特征。文献[51]提出一种减少预测方向候选集数目的算法，对于帧内 16x16 块，所有预测方向均进行 RDO 决策，得到 16x16 模式下的最优预测方向与对应预测方向下的代价 R，再根据 R 与阈值之间的关系来判断 4x4 块中需要进行决策的预测方向的数目，若 R 小于某阈值，则 4x4 块中只有 DC、水平及垂直这三种预测方向需进行 RDO 决策，否则帧内 4x4 块的所有 9 种预测方向均进行 RDO 决策。由于该文献并未使用 MPEG-2 的码流信息，因此，它只是一种 H.264/AVC 的帧内快速模式决策算法。文献[50][51]提出的帧内快速转码算法均为像素域上的转码，文献[52]提出一种在变换域上的帧内快速转码方法，其转码框架图如图 2.8 所示。由于 MPEG-2 与 H.264

的变换方式不同，该文献还提出一种 MPEG-2 DCT 系数转换成 H.264 变换系数的方法。

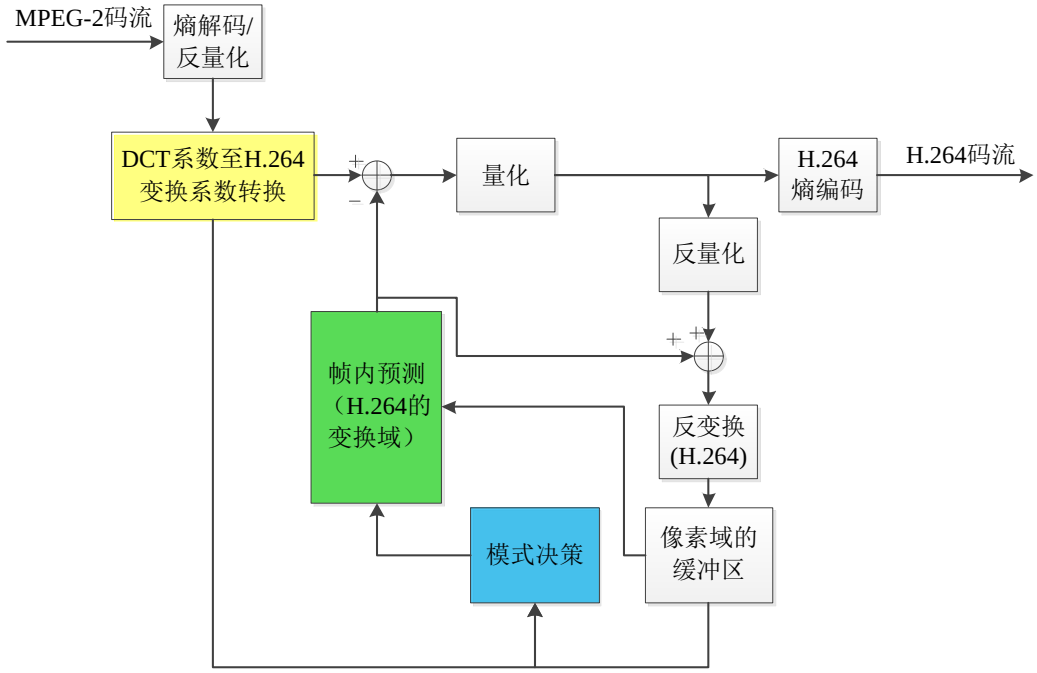


图 2.8 MPEG-2 至 H.264 转码中帧内变换域转码框架图

帧间快速转码一般分成帧间快速模式决策算法[51, 53, 54]及帧间快速运动估计算法[51, 53]。帧间快速模式决策算法快速决策当前 H.264 宏块的划分方式，即只划分成 1 个 16x16 块还是 2 个 16x8 块还是 2 个 8x16 块还是 4 个 8x8 块，对于 8x8 块的划分，递归使用。文献[53]利用帧间模式的代价单调性假设进行快速模式决策，即如果某个小块划分(16x8、8x16 或 8x8)的代价总和比大块(16x16)大，那么其它未进行的小块划分模式可不继续进行，模式决策过程结束。文献[54]提出一种机器学习的方法来进行快速模式决策。它记录 MPEG-2 残差的方差信息、中值信息、H.264 的最优块模式信息，并通过机器学习的方法得到它们间的相关性，即一棵决策树，在进行快速转码时，利用 MPEG-2 残差的方差信息、中值信息与决策树来快速决策当前宏块的模式。文献[51]提出一种利用 16x16 块的 direct 模式的代价与阈值的关系进行快速模式决策的算法。对于帧间快速运动估计算法，一般利用解码的 MPEG-2 的运动矢量做为当前编码块的搜索中心点，再在中心点进行小范围的运动估计。这种方式可以减少搜索点的数目，从而加速运动估计模块。由于 H.264/AVC 引入了多帧参考技术，其它参考帧的搜索中心点可通过将 MPEG-2 的运动矢量进行

伸缩获得。

2.3.2 H.264 与 AVS 的相互转码技术

AVS(Audio and Video Coding standard)[55-56]是由中国音视频编码标准工作组制定而成，我国具备自主知识产权的视频编码标准。由于 AVS 的编码压缩效率[57-58]与 H.264/AVC 相当且它解决了 H.264/AVC 的专利问题，拥有广泛的应用价值，因此，它与 H.264 编码标准均被广泛应用，从而导致了分别使用 H.264 标准和 AVS 标准的多媒体设备之间的不兼容问题。为解决这个问题，AVS 与 H.264/AVC 的相互转码技术得以发展。在进行 AVS 至 H.264/AVC 或者 H.264/AVC 至 AVS 的转码前，必须先进行输入码流与输出码流 QP 的映射，即将 AVS 码流的 QP 映射成 H.264 码流的 QP 或者相反。这是由于 H.264 的量化级别与 AVS 并不相同，H.264 的 QP 范围是 0-51，而 AVS 的 QP 范围是 0-63，若使用相同的 QP，则转码输出的码流大小将与输入大小不同。式（2.1）和式（2.2）分别是 AVS 与 H.264/AVC 的 *Qstep* 计算方法[59] ，为使输出码流大小与输入码流大小一致，应尽可能缩小两者 *Qstep* 的距离，即如果输入的 AVS 码流的 QP 为 28，H.264/AVC 的 QP 应选择 25，而非 24 或者 26。下面我们分别就 AVS 至 H.264 的快速转码算法与 H.264 至 AVS 的快速转码算法进行介绍。

$$Qstep_{avs} \approx (2^{(QP_{avs}//8)}) * QP2QSTEP_{avs}[QP_{avs}\%8] \tag{2.1}$$

$$Qstep_{264} = (2^{(QP_{264}//6)}) * QP2QSTEP_{264}[QP_{264}\%6]) \tag{2.2}$$

$$\mathrm{QP2QSTEP_{avs}} = \{1.0, 1.0905, 1.189, 1.297, 1.414, 1.542, 1.682, 1.834\} \tag{2.3}$$

$$QP2QSTEP_{264} = \{0.625, 0.6875, 0.8125, 0.875, 1, 1.125\} \tag{2.4}$$

$$QP_{\mathrm{avs}} // 8 = \lfloor QP_{avs} / 8 \rfloor \tag{2.5}$$

$$QP_{264} // 6 = \lfloor QP_{264} / 6 \rfloor \tag{2.6}$$

文献[60]提出一种 AVS 至 H.264 的帧内转码算法，由于 AVS 只包含帧内 8x8 块，且只有 5 种方向预测模式，因此，无法直接用一个 AVS8x8 块的帧内模式来决策 H.264 帧内宏块的块大小与方向预测模式，但可联合使用四个 AVS8x8 块的帧内模式来减少 H.264 的帧内方向预测模式的候选集，从而达到加速的目的。文献[59]进一步使用 AVS 的 DCT 系数来进行 H.264 帧内块划分

决策，它将 AVS 的帧内 8x8 块的 DCT 系数分成四类：UP_ROW_NOT_ZERO、NOT_REGULAR_COEFFICIENTS 、 LEFT_COLUMN_NOT_ZERO 及 DC_ALL_ZERO。如果宏块内的四个 8x8 块的 DCT 系数类型均相同，则当前宏块被编码成一个 16x16 块，如果四个 8x8 块 DCT 系数类型均为 NOT_REGULAR_COEFFICIENTS 类型，则被编码成 16 个 4x4 块，其它情况均被编码成 4 个 8x8 块。对于 AVS 至 H.264 的帧间快速转码，文献[61]提出一种基于纹理的模式判别方法，对于 AVS 码流中的 16x16、16x8 及 8x16 块直接映射成 H.264 中对应的块模式，对于 H.264 的 8x8 及 8x8 块的子划分模式将根据 AVS 码流的纹理信息进行模式决策。此外，文献[59]还进一步分析了重用 AVS 的运动矢量信息对转码性能的影响，同时还提出了针对 Skip 模式的 AVS 至 H.264 的帧间转码算法。

文献[62]提出一种 H.264 码流至 AVS 码流的快速转码算法。对于帧内快速转码，它根据 H.264 码流的 SAD 值来计算当前编码 AVS 块的帧内预测方向加入预测方向候选集中，再将 DC 模式、当前编码块的左边块的帧内预测方向这两种模式也加入当前预测块的预测方向候选集，三种模式进行 RDO 得到最优帧内预测方向。在 H.264 至 AVS 的帧间快速转码中，H.264 的模式及运动矢量被直接重用，但由于 AVS 的最小块是 8x8 块，而 H.264 的最小块为 4x4 块，对于 H.264 的块模式为 4x4、8x4 及 4x8 的码流，文献[62]提出一种利用 SAD 值来计算当前 AVS 编码块的运动矢量的算法。

2.3.3 H.264 至 HEVC 的转码技术

H.264/AVC 至 HEVC 的快速转码技术主要分成三类：CTU 四叉树裁剪算法[63-70]、PU 预测技术[63-66,71]及快速运动估计技术[63-65,71]。CTU 的四叉树裁剪算法利用 H.264 的宏块类型信息或者其他解码信息对当前 CTU 的四叉树所有可能情况进行裁剪，从而缩短四叉树递归过程，加速编码过程。PU 预测技术是指利用解码 H.264 信息预测当前编码 CU 的可能预测模式，它一般分成两类：直接预测当前 CU 的 PU 类型及缩减当前 CU 的 PU 候选集数目。快速运动估计算法与一般的标准间快速转码算法一样，它利用 H.264 的运动矢量信息作为当前 PU 的初始搜索点，并在搜索点附近进行小范围的搜索，从而加速编码部分的运动估计模块。

文献[64]提出一种基于 PS-RDO[72]模型的快速转码算法，它利用 PS-RDO 模型预测帧间 CTU 的最优四叉树划分及最优的 PU 模式。对于帧内 CTU，它

利用 H.264 的块模式信息初始化 CTU 的划分，再依次对初始化的 CTU 四叉树中的每个 CU 进行合并，得到每个 CU 的划分深度范围。每个 CU 只需在对应深度范围内进行 RDO，而不再需要进行完整四叉树递归 RDO 过程。文献[65]提出一种离线阈值训练方式，在离线训练时，训练序列进行完整的转码，并记录每个 CU 的最终划分决策与对应的特征值。完整转码后，利用记录的信息得到两个阈值用于快速转码。在进行快速转码时，将当前 CU 的特征值与阈值相比较，从而决策当前 CU 是否需要进一步划分。文献[66]将特征值进行扩展，并将离线训练阈值的方式修改成在线训练，从而使得阈值更自适应，可针对不同内容的视频得到不同的阈值，提升快速转码的准确性。此外，文献[66]还引入了一种同时利用多种特征值的快速转码算法，称为线性判别式法。文献[68][70]将机器学习算法引入到 H.264 至 HEVC 的快速转码研究中，这种机器学习的方法与离线训练方法类似，也需要先进行离线的完整转码与记录过程，差别在于该方法利用机器训练得到 CU 特征值与 CU 是否划分之间的关系。文献[63]首次将率失真模型引入到 H.264 至 HEVC 的快速转码中，它利用 H.264 的宏块码率与对应 H.264 宏块的复杂度的关系将当前 CU 划分成三类，通过判断当前 CU 的复杂度来进行当前 CU 需要进行 RDO 的深度范围。

在 H.264 至 HEVC 的 PU 预测技术研究上，文献[71]提出一种关于 PU 模式预测的方法，它利用 H.264 码流中的运动矢量信息、残差信息及模式信息进行当前深度 PU 模式的预测。若当前 CU 对应的各个 H.264 块的运动矢量均相同，则当前 CU 的 PU 只进行 Merge 及 2Nx2N 模式的决策。接着，该方法还利用当前 CU 对应的各个 H.264 块的运动矢量信息，来跳过 2NxN 模式或者 Nx2N 模式。对于非对称的预测模式，该方法利用当前 CU 对应的 H.264 残差信息进行预测，若 2NxN 上边块的残差大于下边块的残差，那么 2NxnD 模式被跳过，否则，2NxnU 模式被跳过，若 Nx2N 左边块的残差大于右边块的残差，那么 nRx2N 模式被跳过，否则，nLx2N 模式被跳过。对于帧内 PU 模式，只有在当前 CU 不包含任何 H.264 帧内块时，才被跳过。文献[63]也提出一种利用 H.264 的运动矢量信息来减少当前 CU 的 PU 候选集的方法。此外，文献[64]则利用 PS-RDO 模型直接进行当前 CU 的 PU 预测。

在快速运动估计算法研究上，文献[71]提出一种利用当前 CU 包含的 H.264 运动矢量信息自适应的调整运动估计时的搜索范围的算法以加速运动估计模块。文献[63]也提出了类似的快速运动估计算法。

除了上述三类 H.264 至 HEVC 的快速转码算法外，文献[67]首次将 WPP 并行算法、SIMD 的加速与快速转码算法相结合，提出了一种极快的 H.264 至 HEVC 的快速转码算法。由于使用了并行及 SIMD 加速，转码速度得到了很大的提升。此外，文献[73]还提出了一种针对监控视频的快速转码方案，该方案针对 H.264 的监控档次码流至 HEVC 码流的转码而提出。

2.4 本章小结

本章就目前标准内及标准间的转码技术的研究现状进行阐述及分析。早期的视频转码技术的研究点主要集中在降码率、降分辨率及降帧率的快速转码中，但随着互联网及软、硬件处理能力的增加，分辨率、码率及帧率早已不是多媒体设备间交互的障碍。由于视频编码标准不断涌现，视频转码技术目前主要集中在标准之间的转码研究上。随着 HEVC 编码标准的诞生及使用，目前市场上主流的 H.264/AVC 编码标准至 HEVC 编码标准的快速转码研究已成为视频转码领域的新热点。本文也主要针对 H.264/AVC 至 HEVC 的快速转码算法进行研究，并提出了几套转码解决方案。

第 3 章 基于 AMV-RDO 的快速帧间转码算法

3.1 引言

文献[20]中提到，HEVC 的编码复杂度在最坏情况下较 H.264 提升了 5 倍。因此，对于 H.264 至 HEVC 的转码应用，其最大难点在于由 HEVC 的高复杂度而带来的转码速率低的问题。如何进行快速的 H.264 至 HEVC 的转码，对于实时应用来说尤为重要。文献[20]中介绍，对于 HEVC 的帧间编码来说，其最大编码复杂度来源于编码树单元(coding tree unit, CTU)的四叉树递归划分决策及由 PU 候选数目增加而引入的 PU 模式决策。针对于这两个高复杂度的模块，本文提出一种基于 AMV-RDO(Avergae Motion Vector Rate Distortion Optimization)率失真模型的快速帧间转码算法。AMV-RDO 利用 H.264 码流的解码运动矢量信息来进行 PU 及 CU 的率失真建模，从而取代 HEVC 的率失真模型来决策 CTU 的四叉树划分并对每个 CU 的 PU 模式进行预测。利用 AMV-RDO 决策出的 CTU 四叉树划分结构即为当前 CTU 的最终四叉树划分结构，不再进行改变，但预测的 PU 并非当前编码 CU 的最终选择 PU，而是与合并模式(Merge Mode)、可选的帧内模式(Intra Mode)一起加入当前 CU 的 PU 候选集中，通过 HEVC 的原率失真模型进行最终的模式决策。由于 AMV-RDO 率失真模型省去了复杂度极高的运动估计过程及变换四叉树划分决策过程及解码过程，因此，它比 HEVC 的原率失真模型更快速，实验结果表明，本算法在带来较小的 RD 性能损失的同时，编码加速比达到 8 倍，即相比于全解全编的转码算法，该算法平均节约了 87%的转码时间。因而，该算法十分适应于具有实时需求的快速转码应用。

下面就该算法的整体算法框架、详细算法描述及实验结果进行介绍，并在本章的最后进行小结。

3.2 快速转码框架

图 3.1 是基于 AMV-RDO 的快速转码的整体框架图。它包括两部分：解码部分和编码部分。在解码部分，输入的 H.264 码流通过熵解码获得预测残差系数、帧内或者帧间的预测信息、模式信息等。预测残差系数通过反量化

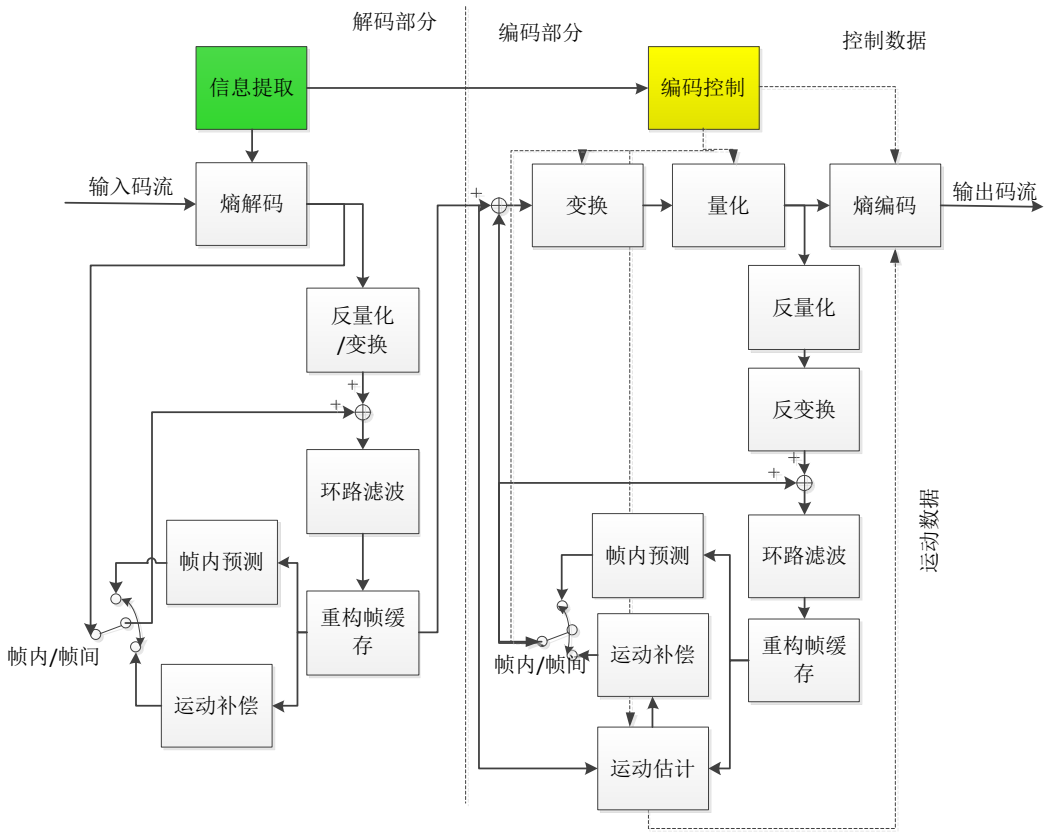


图 3.1 基于四叉树预测的帧间快速转码框架图

和反变换得到预测残差，帧内或者帧间的预测信息、模式信息作用于帧内预测或者运动补偿模块得到当前解码块的预测块。再将预测块与预测残差相加，即得到重构块。重构块进行环路滤环后，放入重构帧缓冲区。至此，解码部分完成。相比于一般的解码器，该快速转码框架中的解码部分新增加了信息提取模块，该模块将熵解码后的 H.264 信息提取出来，放入固定的内存空间中，在解码完成后，随同解码 YUV 重构帧一起送入编码部分。在本算法中，信息提取模块只需提取帧间宏块的运动矢量信息及模式信息，并将运动矢量信息以 4x4 块为单位存储到一个二维数组空间中。提取的解码信息将用于编码部分的编码控制模块，以进行 AMV-RDO 及 HEVC 的默认 RDO 过程。在编码部分，将解码 YUV 重构帧划分成一个个 CTU 输入到编码部分， CTU 又通过编码控制单元递归划分成 CU，CU 是实际的编码单位。对于每个 CU，编码控制单元判断其所属帧类型以进行当前 CU 的预测过程，若当前 CU 属于 I 帧，则只需进行帧内的模式决策及预测方向决策，从而获得当前 CU 的预测块；若当前 CU 属于 B、P 帧，则既需进行帧内的模式决策及预测方向决策，

获得对应的帧内预测块，还需进行帧间的模式决策及运动估计与运动补偿操作，获得对应的帧间预测块，最终预测块需对比两种预测模式下的 RD(rate distortion)值来决定。通过编码控制单元获得当前 CU 的最优预测块后，通过将 CU 与预测块相减，得到当前 CU 的预测残差，预测残差再进行变换、量化操作转换成残差系数，最后，残差系数经过熵编码成为 HEVC 码流。

在H.264至HEVC的转码中，编码端最复杂的模块当属编码控制模块，它有点类似于人体的大脑，通过率失真优化(rate distortion optimization, RDO)机制控制整个编码过程，包括：CTU的四叉树递归划分过程，预测过程及变换四叉树递归划分过程。对于CTU的四叉树递归划分过程，编码控制模块决定四叉树中的每个CU节点是否需要进一步划分成四个子CU。对于帧内预测，它决定当前CU的最优PU模式及最优PU模式下对应的预测方向模式。对于帧间预测，它决定当前CU的最优帧间模式及运动估计的搜索范围。对于变换四叉树递归划分过程，它将预测残差划分成大小不同的块以进行相应大小的变换过程，从而尽可能消除预测残差的空间相关性。因此，在快速转码中，我们通常利用解码信息作用于该模块进行编码部分的加速，在本算法中，我们利用解码的运动矢量信息及模式信息作用于该模块，并引入AMV-RDO模型以代替HEVC的原RDO模型决策CTU的四叉树递归划分及PU预测，对于CU下的PU决策仍使用HEVC的原RDO模型。

3.3 算法描述

本算法主要利用 H.264 解码码流中的运动矢量信息及模式信息来修改 HEVC 编码器中的编码控制模块。除 HEVC 本身的 RDO 模型(Original RDO, O-RDO)，我们在该模块中还引入了 AMV-RDO 模型。利用 AMV-RDO 模型来进行当前 CTU 四叉树的划分决策与 PU 预测，接着以深度优先顺序依次访问划分好的 CTU 四叉树的各个叶子结点 CU，将预测 PU、Merge 模式与可选的 intra 模式加入该 CU 的 PU 候选集，利用 O-RDO 模型对 PU 候选集中的所有 PU 进行决策，从而得到当前 CU 的最优 PU 模式及相应的预测信息、残差信息，最后按 AMV-RDO 决策出的 CTU 四叉树结构的深度优先顺序将各决策结果依次编入码流形成 HEVC 码流。因此，该算法总共分成两部分内容，第一部分利用 AMV-RDO 模型决策当前 CTU 的四叉树划分及预测 PU，第二部分利用 O-RDO 模型决策 CTU 四叉树各个叶子结点的 PU 模式。下面，本节就该算法进行详细描述。

3.3.1 O-RDO 模型下的 PU 决策及 CTU 四叉树划分决策

RDO 是一种进行模式决策的工具，它求解各个模式下对应的 RD 值以决策出最优的模式。RD 值通过式（3.1）计算，通常 R 指当前模式下编码需要的码率，D 指失真，即当前模式下的重构图像与原始图像的差异，D 的计算方式有很多，一般为 SAD、SSD 及 SATD 等[74]。λ 是调节 D 值与 R 值的系数，通常与编码 QP 有关。

在 O-RDO 模型中，PU 决策需要进行完整的编码流程及解码流程，以得到当前模式下的重构块及编码消耗码率，即在当前 PU 模式下需进行完整的运动估计、运动补偿操作以得到预测块，通过将原始块与预测块相减得到预测残差，预测残差经过变换四叉树的递归划分决策及量化操作最终进行熵编码以计算当前 PU 模式下的 R 值，量化后的系数通过反量化、反变换及帧内或帧间的预测得到重构图像以计算对应 D 值，通过式（3.1）即可得到当前 PU 模式下的 RD 值。最优 PU 模式通过式（3.2）获得，其中 PU_{best} 指当前 CU 下的最优 PU，RD_{PU_i} 指第 i 个 PU 模式下的 RD 值，因此，式（3.2）可以解读为所有 PU 模式下 RD 值最小的 PU 被选为当前 CU 下的最优 PU。

$$RD_{value} = D + \lambda R \tag{3.1}$$

$$PU_{best} = \arg\min_{PU_i} RD_{PU_i} \tag{3.2}$$

对于 CU 的划分决策，当前 CU 下的最优 PU 的 RD 值暂且作为当前 CU 的 RD 值，接着将当前 CU 划分成四个子 CU，依次计算其 RD 值并加和，将加和后的 RD 值与当前 CU 的 RD 值比较，若前者小，则当前 CU 划分，且将子 CU 的 RD 和更新为当前 CU 的 RD 值，否则当前 CU 不划分，当前 CU 的 RD 值仍为最优 PU 的 RD 值。CTU 的四叉树划分决策必须自底向上进行，因为若子 CU 未进行划分决策，则子 CU 的最终 RD 值并不可知。

3.3.2 AMV-RDO 模型下的 PU 预测

AMV-RDO 模型下的 PU 预测与 O-RDO 模型下的 PU 决策类似，它通过比较所有 PU 模式的 *RD* 值来预测当前 CU 最可能的 PU 模式。与 O-RDO 不同的是，AMV-RDO 模型下的 *RD* 值计算并不需要进行完整的编码过程与解码过程，它利用 H.264 解码的运动矢量信息直接进行当前 PU 模式下的运动矢量的估计而不需运动估计搜索获得。预测的运动矢量 MV_{pred} 可通过式（3.3）计算获得，其中 $MV_{pred\text{-}i}$ 指第 i 个 PU 模式下的预测运动矢量，N_i 指当前 PU

块所对应的 H.264 块所包含的运动矢量的个数，由于 H.264 的运动矢量以 4x4 块为单位进行存储，因此对于 MxN 大小的 PU 块，其 N_k 值可以直接通过式（3.4）计算得到，MV_k 指当前 PU 块所对应的 H.264 块的第 k 个运动矢量。将估计得到的运动矢量进行运动补偿从而得到当前 PU 模式下的预测块，并计算预测块与原始图像的差异即可得到 D 值，由于该模式下的运动矢量是通过估计获得，因此，我们在计算其 RD 值时，不考虑码流的大小，直接将 D 值作为当前模式下的近似 RD 值，即式（3.6），其中 $RD_{AMV\text{-}i}$ 指 AMV-RDO 模型下的第 i 个 PU 模式的 RD 值，D_i 指 AMV-RDO 模型下第 i 个 PU 模式的 D 值。需要注意的是在进行 PU 下的运动矢量估算时， 它利用的 H.264 运动矢量均指缩放后的运动矢量值，其具体缩放过程可参见 3.3.3 节。

$$MV_{\text{pred}-i} = \frac{1}{N_i}\sum_{k=1}^{N_i} MV_k \tag{3.4}$$

$$N_{\text{k}} = (M/4)\times(N/4) \tag{3.5}$$

$$RD_{AMV-i} \approx D_i \tag{3.6}$$

由于在 AMV-RDO 模型下，PU 模式的决策省去了耗时的运动估计过程，并且由于直接用预测块作为重构块来求解 D 值，还省去了变换、量化及解码过程，因此，AMV-RDO 模型下的 PU 决策较 O-RDO 模型下的 PU 决策更快速。在本算法中，我们只进行 PART_2Nx2N，PART_2NxN，PART_Nx2N，PART_2NxnU，PART_2NxnD，PART_nLx2N，PART_nRx2N 及 PART_NxN 这八种模式的 AMV-RDO 决策，即预测 PU 只可能是这八种 PU 模式之一。对于 Merge 模式，它不加入 PU 预测的范围。此外，由于 PART_NxN 模式只有在当前 CU 是最小 CU 时才有效，因此，一般情况下，该模式也不加入决策范围。

3.3.3 AMV-RDO 模型下的解码运动矢量的缩放

由于 H.264 中已引入多参考帧技术，因此，H.264 中的各个运动矢量的参考帧可能各不相同，利用参考帧各不相同的运动矢量进行式（3-4）的计算来求取当前 PU 的估算运动矢量是无意义的，因此，在进行估计之前必须先将所有的 H.264 运动矢量进行缩放，即利用式（3.7）将原始解码运动矢量缩放到同一参考帧中。 其中 MV_{scale} 指缩放后的运动矢量，POC_{cur} 指当前编码帧的 POC 值，POC_{ref} 指当前运动矢量的参考帧的 POC 值，MV_{ori} 即原始解码的

运动矢量值，对于 P 帧及 B 帧前向，POC_{scale} 指当前编码帧的前一参考帧的 POC，对于 B 帧后向，POC_{scale} 指当前编码帧的后一参考帧的 POC。

$$MV_{scale} = \frac{POC_{cur} - POC_{scale}}{POC_{cur} - POC_{ref}} MV_{ori} \tag{3.7}$$

3.3.4 AMV-RDO 下的 CTU 四叉树划分预测

AMV-RDO 下的 CTU 四叉树划分过程与 O-RDO 下的 CTU 四叉树划分过程一样，通过比较当前 CU 的 *RD* 值与四个子 CU 的 *RD* 值之和来决策当前 CU 是否需要划分。不同的是，AMV-RDO 下的 *RD* 值通过 3.3.2 所描述的方式求解得到。此外，AMV-RDO 下的 CTU 四叉树划分过程也必须是自底向上进行，否则子 CU 的 *RD* 值并非其最终 *RD* 值。

3.3.5 算法流程

图 3.2 为该算法的整体算法流程，该算法整体分成两部分，第一部分利用 H.264 运动矢量进行当前 CTU 的四叉树划分决策与 PU 的预测，第二部分以深度优先顺序依次访问决策好的 CTU 四叉树的各个叶子结点，并利用 H.264 的块模式信息进行叶子结点 CU 的 PU 决策，以得到最优 PU 及对应的预测信息、残差信息及重构图像。需要注意的是第一部分利用的 H.264 运动矢量均指缩放后的运动矢量信息。

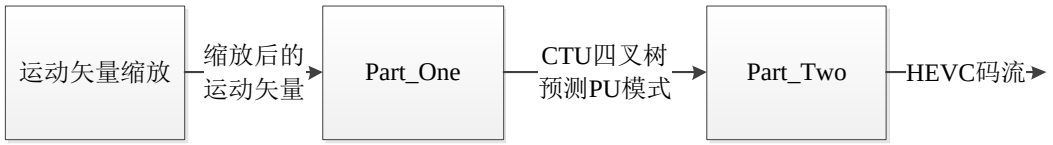


图 3.2 基于 AMV-RDO 的快速转码算法的整体流程图

第一部分的算法流程如图 3.3 所示，依次为：

Step 1.1: 利用式（3.4）计算当前 PU 模式下的运动矢量 MVpred-i

Step 1.2: 利用 MVpred-i 进行运动补偿，获取当前 CU 的预测块

Step 1.3: 计算当前模式下的 RD 值

Step 1.4: 利用 AMV-RDO 模型对当前 CU 进行 PU 的预测，得到预测 PU 及对应 RD 值

Step 1.5: 将当前 CU 划分成四个子 CU（SubCU）

Step 1.6: 重复 Step 1.1 至 Step 1.4，获得每个 SubCU 的预测 PU 模式，及对应的 RD 值

Step 1.7: 利用 AMV-RDO 模型进行当前 CU 的划分决策

Step 1.8: 自底向上进行 CU 的划分决策，得到当前 CTU 的四叉树划分结构

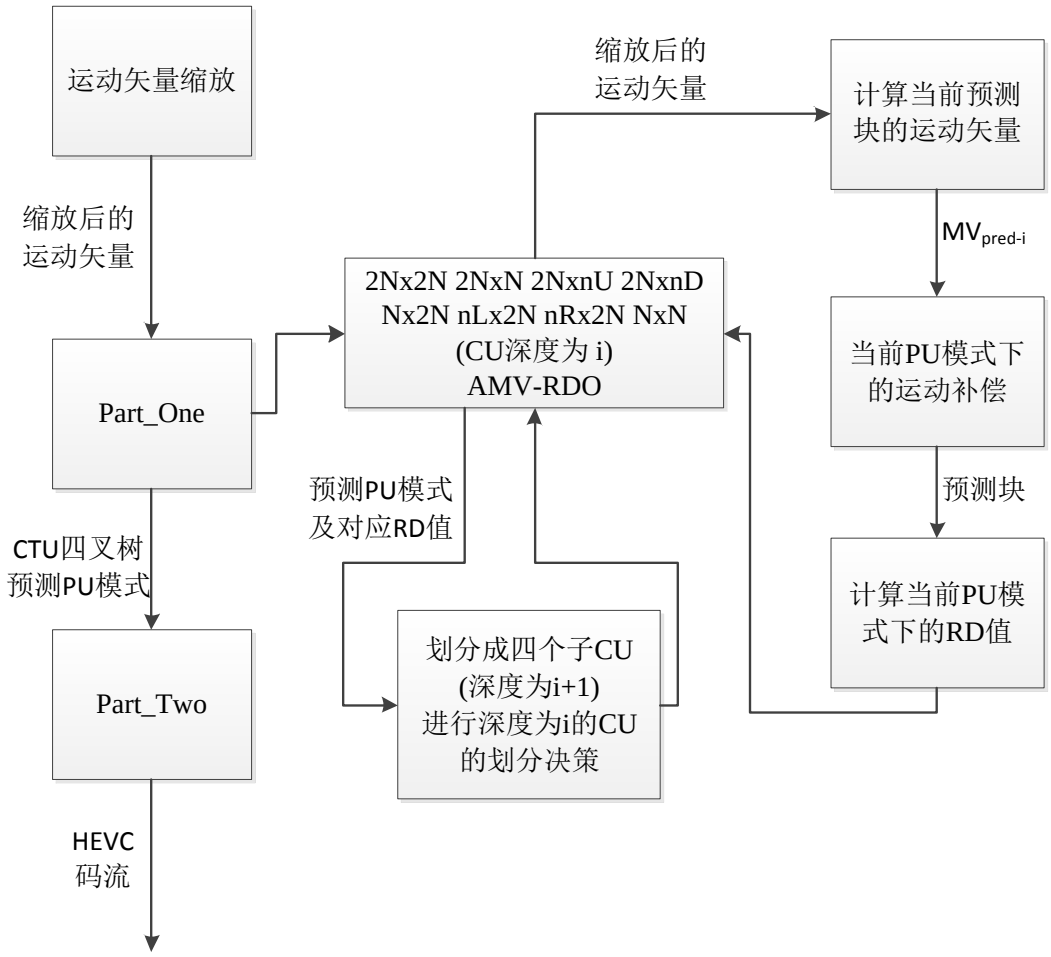


图 3.3 AMV-RDO 下的四叉树划分决策与 PU 预测流程图

第二部分利用 O-RDO 模型进行 CU 的 PU 决策，其 PU 候选集包括 Merge 模式、预测 PU 模式及可选的 intra 模式。intra 模式根据当前 CU 是否包含 H.264 intra 块可选的加入 PU 候选集，若当前 CU 所对应的 H.264 块不包含 intra 模式，则 HEVC 的 intra 模式不加入 PU 的候选集，否则加入。图 3.4 是 AMV-RDO 下的 PU 决策流程图。

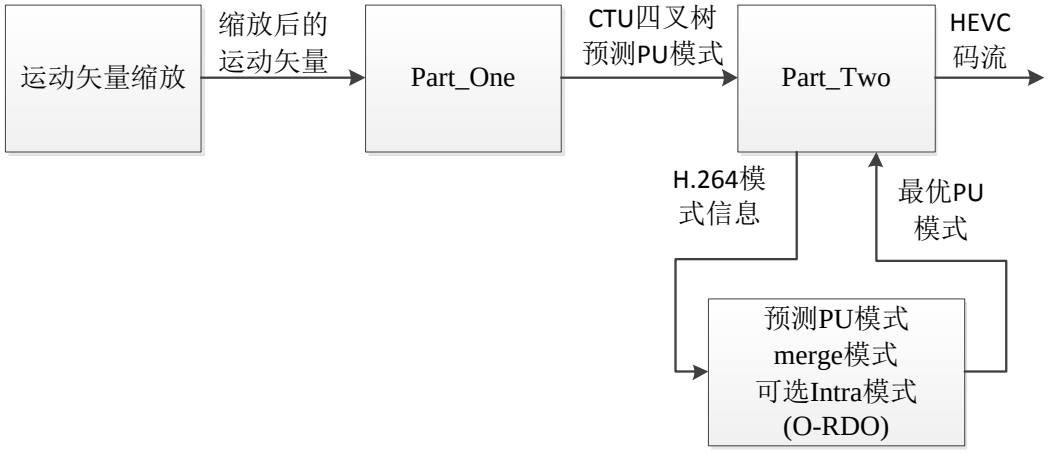


图 3.4 PU 决策流程图

3.4 实验结果

3.4.1 编码配置

所有的源 H.264 码流均采用 JM18.6[75]参考软件下的 high profile 档次编码获得，编码配置文件采用 LDB，码流结构为 IBBPBBPBBP，其中 I 帧 QP 为 20，B、P 帧 QP 均为 21。分别编码分辨率为 352x288、416x240、832x480 及 720P 下的 H.264 码流共 14 个，详见表 3.1。对于全解全编的转码器，我们利用开源软件 ffmpeg 2.0 实现，ffmpeg 2.0 包含多个标准的解码器与编码器，是目前市场上主流的转码应用系统。因此，我们直接利用 ffmpeg 2.0 的 H.264/AVC 的解码器进行码流的解码，并将开源软件 multicorex265 的 1.3 版本作为 HEVC 编码库加入 ffmpeg 2.0 中，由此实现自 H.264 至 HEVC 的全解全编转码器。我们的快速转码算法也在该全解全编转码器的基础上进行实现。

3.4.2 实验结果及分析

表 3.2 显示了本算法的 RD 性能及加速比情况，从表 3.2 中可以看出基于 AMV-RDO 的快速转码算法相比于全解全编的转码算法在 CIF 序列、416x240、832x480 及 720P 序列的加速比分别为 7.88、9.30、8.50 及 5.90，平均加速比达到 7.61 倍，即相比于全解全编的转码系统，它可以节约 87%的转码时间。在 RD 性能损失上，基于 AMV-RDO 的快速转码算法较全解全编的转码算法平均损失了 17%的 BD-Rate，即在相同的码流大小下，该算法的图像质量较全解全编的图像质量平均下降 0.7dB。表 3.3 进一步显示了本算法与算法[66]在转码速度与转码质量这两方面的对比结果，从对比结果可以看出，算法[66]

具有更好的转码 RD 性能，但加速比十分有限。而本章所提出的算法在损失一定质量的前提下，极大的提升了转码速度。因此，这两个算法可以适用于需求各不相同的应用场景中。

表 3.1 编码序列及对应分辨率大小

分辨率	序列名称	帧数	帧率
CIF(352x288)	Coastguard	300	30fps
	Container	300	30fps
	Flower	300	30fps
416x240	BasketballPass	500	50fps
	BQSquare	600	60fps
	RaceHorses	300	30fps
832x480	BasketballDrill	500	50fps
	BQMall	600	60fps
	RaceHorses	300	30fps
720P(1280x720)	FourPeople	600	60fps
	Johnny	600	60fps
	KristenAndSara	600	60fps
	Vidyo1	600	60fps
	Vidyo3	600	60fps

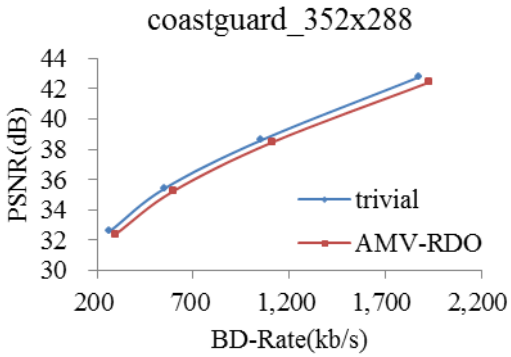
图 3.5 描述了本算法中各个序列的 RD 性能曲线，其中红色曲线代表基于 AMV-RDO 的快速转码算法的 RD 性能，蓝色曲线代表全解全编转码算法的 RD 性能，从图 3.5 可以看出，尽管本章提出的快速转码算法较全解全编的转码算法有一定的性能损失，但其 RD 性能曲线仍十分接近于全解全编转码算法的 RD 曲线。基于 AMV-RDO 的快速转码算法的性能之所以有一定的损失是由于 AMV-RDO 的率失真模型并未进行完全的编码操作及解码操作，直接计算的预测块的运动矢量可能并不准确，从而导致预测块并不准确，继而可能致使 CTU 的四叉树划分结构并不准确，从而引入性能损失。

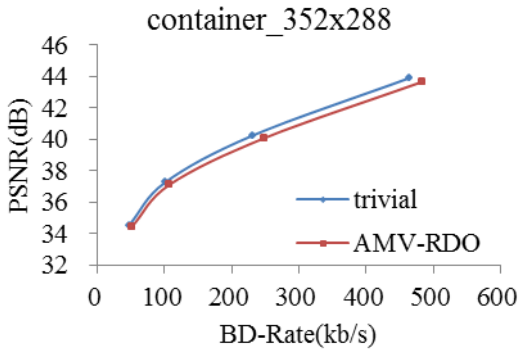
表 3.2 基于 AMV-RDO 的快速转码算法性能

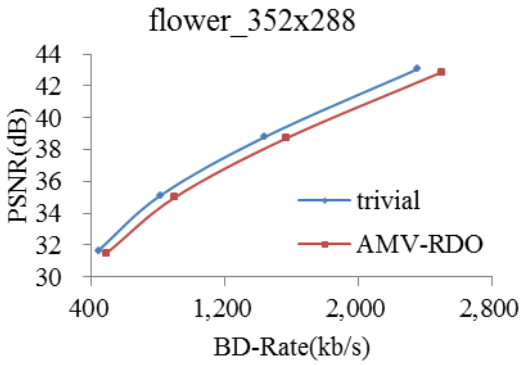
分辨率	序列	全解全编			AMV-RDO 快速转码		
		BD-Rate	BD-PSNR	加速	BD-Rate	BD-PSNR	加速
CIF(352x288)	Coastguard	0.00%	0.00dB	1.00	10.17%	-0.52dB	8.50
	Container	0.00%	0.00dB	1.00	10.98%	-0.42dB	5.96
	Flower	0.00%	0.00dB	1.00	11.30%	-0.73dB	9.18
416x240	BasketballPass	0.00%	0.00dB	1.00	18.44%	-1.01dB	9.52
	BQSquare	0.00%	0.00dB	1.00	14.82%	-0.59dB	8.20
	RaceHorses	0.00%	0.00dB	1.00	19.70%	-1.07dB	10.17
832x480	BasketballDrill	0.00%	0.00dB	1.00	23.32%	-0.99dB	10.25
	BQMall	0.00%	0.00dB	1.00	19.09%	-0.84dB	9.00
	RaceHorses	0.00%	0.00dB	1.00	18.11%	-0.82dB	6.25
720P(1280x720)	FourPeople	0.00%	0.00dB	1.00	18.87%	-0.62dB	6.25
	Johnny	0.00%	0.00dB	1.00	21.69%	-0.49dB	5.00
	KristenAndSara	0.00%	0.00dB	1.00	19.82%	-0.57dB	6.25
	Vidyo1	0.00%	0.00dB	1.00	20.12%	-0.57dB	6.00
	Vidyo3	0.00%	0.00dB	1.00	18.75%	-0.52dB	6.00
CIF(352x288)		0.00%	0.00dB	1.00	15.95%	-0.56dB	7.88
416x240		0.00%	0.00dB	1.00	17.65%	-0.89dB	9.30
832x480		0.00%	0.00dB	1.00	20.17%	-0.88dB	8.50
720P(1280x720)		0.00%	0.00dB	1.00	19.85%	-0.55dB	5.90
整体		0.00%	0.00dB	1.00	17.51%	-0.70dB	7.61

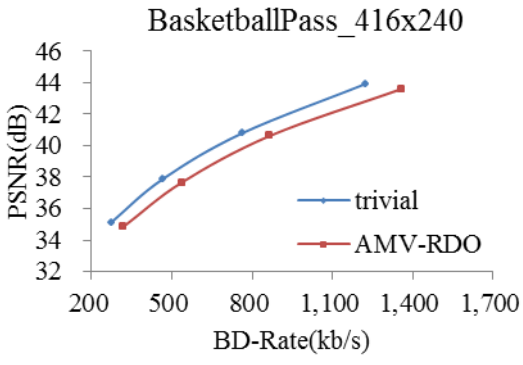
表 3.3 基于 AMV-RDO 的快速转码算法与文献[66]算法性能对比

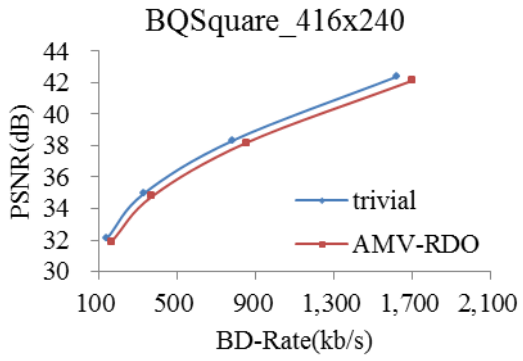
分辨率	序列	文献[66]		AMV-RDO 快速转码	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	4.20%	1.67	10.17%	8.50
	container	4.73%	2.06	10.98%	5.96
	flower	3.46%	1.93	11.30%	9.18
416x240	BasketballPass	5.04%	1.70	18.44%	9.52
	BQSquare	4.99%	2.28	14.82%	8.20
	RaceHorses	4.75%	1.69	19.70%	10.17
832x480	BasketballDrill	5.13%	1.82	23.32%	10.25
	BQMall	5.16%	1.64	19.09%	9.00
	RaceHorses	3.80%	1.83	18.11%	6.25
720P(1280x720)	FourPeople	4.75%	1.62	18.87%	6.25
	Johnny	3.71%	1.50	21.69%	5.00
	KristenAndSara	4.17%	1.42	19.82%	6.25
	Vidyo1	4.97%	1.54	20.12%	6.00
	Vidyo3	5.07%	1.85	18.75%	6.00
CIF(352x288)		4.13%	1.89	15.95%	7.88
416x240		4.93%	1.89	17.65%	9.30
832x480		4.70%	1.76	20.17%	8.50
720P(1280x720)		4.53%	1.59	19.85%	5.90
整体		4.57%	1.75	17.51%	7.61

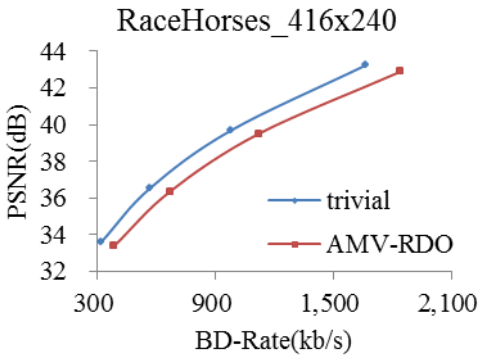


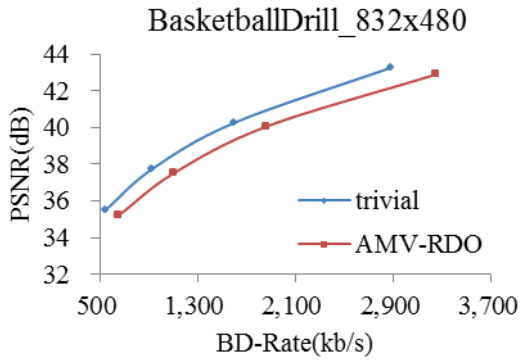


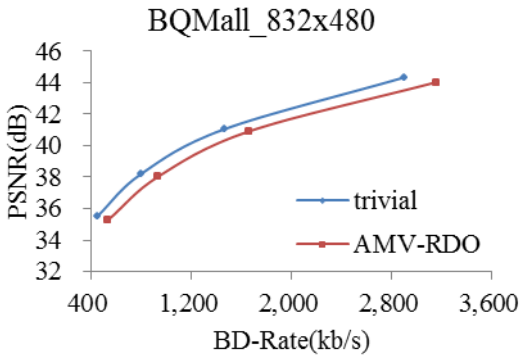

flower_352x288
PSNR(dB)
44
42
40
38
36
34
32
30
400
1,200
2,000
2,800
BD-Rate(kb/s)
trivial
AMV-RDO

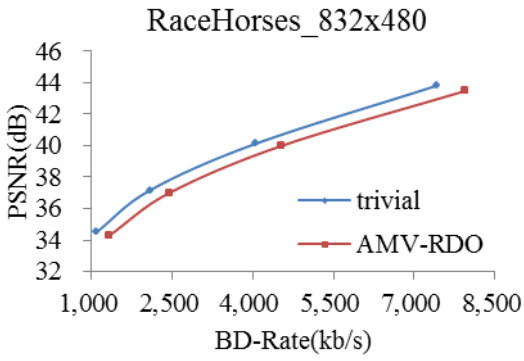

BasketballPass_416x240
PSNR(dB)
46
44
42
40
38
36
34
32
200
500
800
1,100
1,400
1,700
BD-Rate(kb/s)
trivial
AMV-RDO

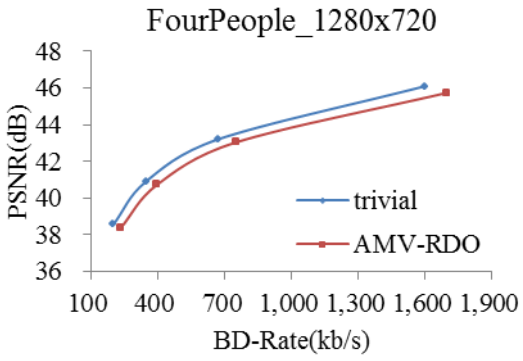

BQSquare_416x240
PSNR(dB)
44
42
40
38
36
34
32
30
100
500
900
1,300
1,700
2,100
BD-Rate(kb/s)
trivial
AMV-RDO


RaceHorses_416x240
PSNR(dB)
44
42
40
38
36
34
32
300
900
1,500
2,100
BD-Rate(kb/s)
trivial
AMV-RDO


BasketballDrill_832x480
PSNR(dB)
44
42
40
38
36
34
32
500
1,300
2,100
2,900
3,700
BD-Rate(kb/s)
trivial
AMV-RDO


BQMall_832x480
PSNR(dB)
46
44
42
40
38
36
34
32
400
1,200
2,000
2,800
3,600
BD-Rate(kb/s)
trivial
AMV-RDO


RaceHorses_832x480
PSNR(dB)
46
44
42
40
38
36
34
32
1,000
2,500
4,000
5,500
7,000
8,500
BD-Rate(kb/s)
trivial
AMV-RDO


FourPeople_1280x720
PSNR(dB)
48
46
44
42
40
38
36
100
400
700
1,000
1,300
1,600
1,900
BD-Rate(kb/s)
trivial
AMV-RDO

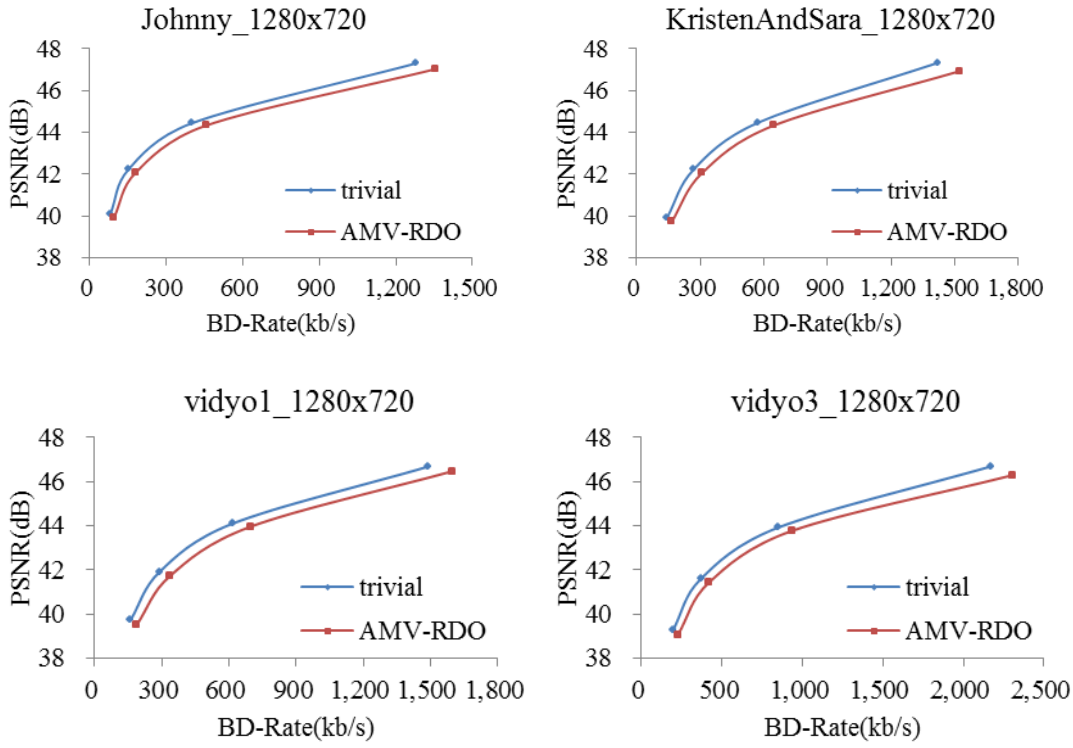


图 3.5 AMV-RDO 快速转码算法与全解全编转码的 RD 曲线对比图

3.5 本章小结

本章提出了一种基于 AMV-RDO 的快速帧间转码算法。该算法利用解码的 H.264 运动矢量信息来进行当前 CTU 的四叉树划分决策与 PU 的预测，并进一步利用 HEVC 的原始率失真模型及 H.264 的模式信息进行 CTU 四叉树叶子结点的 PU 决策，从而生成 HEVC 码流。实验结果表明该算法在引入一定的质量损失的前提下，具有非常高的加速比，相比于全解全编的转码，本算法的加速比达到 7.61 倍，即节省了 87%的转码时间，因此，本算法十分适用于对质量要求并非十分严格，但具有实时要求的转码应用中。

第 4 章 固定阈值下的快速转码算法

4.1 引言

基于 AMV-RDO 的快速转码算法在引入一定质量损失的前提下拥有非常高效的转码速率，因此，十分适用于具有实时要求的转码应用中，然而，由于该算法将带来一定的转码质量损失，因此，并不适用于高质量转码应用中。本章，我们介绍一种固定阈值下的快速转码算法，它利用当前 CU 的特征值与阈值之间的关系进行当前 CU 的划分决策与 PU 的模式选择。实验结果表明，本算法在几乎不带来转码质量损失的同时，可以节省近一倍的转码时间。

下面，我们就阈值计算、快速转码算法、转码算法框架及实验结果进行介绍，并在本章最后进行总结。

4.2 阈值计算

本算法的阈值通过离线训练获得，它利用全解全编转码记录每个 CU 的划分决策与对应的特征值，并在转码完成之后，利用记录得到的 CU 决策信息与特征值信息进行对应的阈值计算。在全解全编转码中，CU 的划分决策记录在数组 C_d 中，对应的特征值记录在数组 F_d 中，其中，d 代表深度，即每个深度的 CU 划分决策及对应的特征值需分别记录下来。当全解全编转码算法完成后，F_d 及 C_d 数组记录完成，其内容分别用式（4.1）及式（4.2）表示，其中 N 代表深度为 d 的 CU 的个数，f_i^d 代表深度 d 下的第 i 个 CU 的特征值，c_i^d 代表深度 d 下的第 i 个 CU 的划分决策，它必须在完整编码 RDO 后才可获取。为简化 C_d 的记录，我们将 CU 的划分决策分成三类：CU 进一步划分成四个子 CU；CU 不继续划分，PU 模式选择 Merge 模式或 PART_2Nx2N 模式；CU 不继续划分，PU 模式选择除 Merge 模式及 PART_2Nx2N 模式外的其它模式，如式（4.3）所示。

$$F_d = \{f_0^d, f_1^d, f_2^d, \cdots, f_{N-1}^d\} \tag{4.1}$$

$$C_d = \{c_0^d, c_1^d, c_2^d, \cdots, c_{N-1}^d\} \tag{4.2}$$

$$c_i^d = \begin{cases} 0 & ,split \\ 1 & ,PU_{mode} = merge \| PART\_2Nx2N \\ 2 & ,others \end{cases} \tag{4.3}$$

数组 F_d 与 C_d 记录完成后，便可利用这两数组计算深度 d 下的阈值：Tlow 与 Thigh。

Tlow 的计算遵循以下三点法则：

1. 特征值 f<=Tlow 的所有 CU 中，c_i^d 值为 1 的比率达到 90%；
2. 任何小于 Tlow 的特征值作为 Tlow 也满足条件 1；
3. 同时满足条件 1 与条件 2 的所有特征值中的最大值被选为 Tlow；

同理，Thigh 也遵循以下三点法则：

1. 特征值 f>=Thigh 的所有 CU 中，c_i^d 值为 0 的比率达到 90%；
2. 任何大于 Thigh 的值作为 Thigh 也满足条件 1；
3. 同时满足条件 1 与条件 2 的所有特征值中的最大值被选为 Thigh；

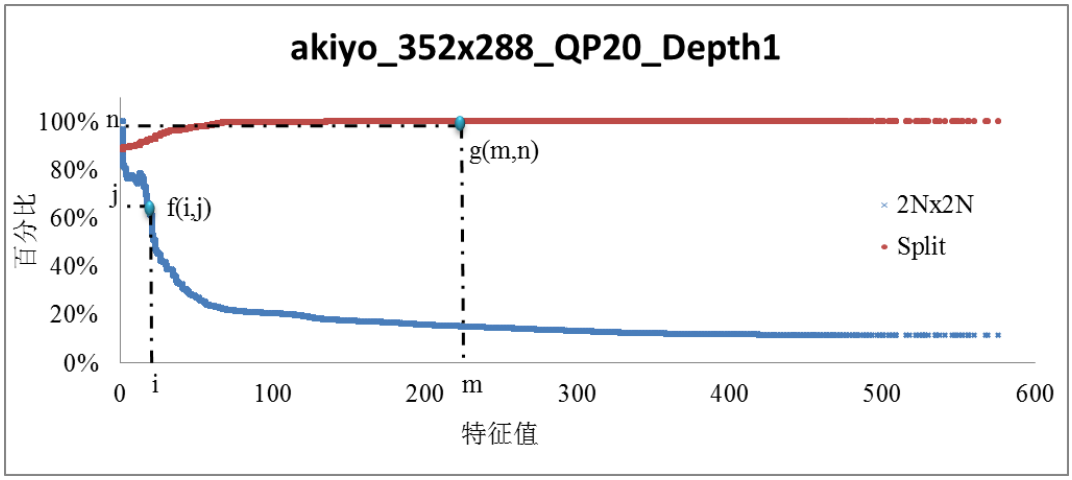


图 4.1 akiyo 序列在 QP=20、depth=1 下的 Fd 与 Cd 关系曲线图

Tlow 与 Thigh 计算法则中的 90%概率只是一个启发性数字，可以通过修改该数值以控制快速转码的复杂度。图 4.1 是 akiyo 序列在 QP 为 20，深度为 1 下记录的 F_d 与 C_d 的关系曲线图，其中，蓝色曲线上的点 f(i,j)代表特征值小于或等于 i 的所有 CU 中，c_i^d 值为 1 的比率为 j。红色曲线上的点 g(m,n)代表特征值大于或等于 m 的所有 CU 中，c_i^d 值为 0 的比率为 n。从图 4.1 中可以观察到，一般特征值越小，当前 CU 不进一步划分且 PU 最终模式选择 Merge 模式或者 PART_2Nx2N 模式的概率越大；特征值越大，当前 CU 进一步划分成 4 个子 CU 的概率也越大。

4.3 快速转码算法

固定阈值下的快速转码算法只应用于深度小于 2 的 CU 的划分决策与 PU 选择，对于深度大于或等于 2 的 CU，由于 H.264 的块大小与 CU 大小相同，我们直接利用 H.264 块类型与 CU 划分之间的统计关系进行映射。下面，分别就帧内快速转码算法与帧间快速转码算法进行介绍。

4.3.1 帧内快速转码算法

由于帧内转码并不包含解码运动矢量信息，因此，帧内快速转码算法只利用两个特征值：DCT 非零系数的数目及 DCT 系数能量。DCT 非零系数的数目由式（4.4）计算获得，其中 C_i 代表 H.264 解码块的 DCT 系数，当该系数不为零时，特征值加 1，因此，DCT 非零系数的数目只计算 H.264 的 DCT 块中系数不为零的数目，而不考虑具体系数的大小。DCT 系数能量由式（4.5）计算所得，它与 DCT 非零系数的数目不同之处在于，该特征值考虑每个系数的大小。

$$E_N = \sum_{\mathrm{i}} M_i$$
$$M_{\mathrm{i}} = \begin{cases} 1 & , C_i \neq 0 \\ 0 & , C_i = 0 \end{cases} \tag{4.4}$$

$$E_C = \sum_{\mathrm{i}} C_i^{\,2} \tag{4.5}$$

在利用 4.2 节介绍的阈值计算方法获取帧内阈值 Tlow 与 Thigh 后，便可利用该阈值进行帧内快速转码。对于深度为 0 和 1 的 CU，我们利用当前 CU 的特征值 f 与阈值之间的关系进行当前 CU 的划分决策，具体为：

1. 如果 f<=Tlow，只进行当前 depth 下的 PART_2Nx2N 模式，并且当前 CU 不继续划分。
2. 如果 f>=Thigh，当前 CU 直接划分成四个子 CU。
3. 如果 f 介于 Tlow 与 Thigh 之间，进行当前 depth 下的 PART_2Nx2N 模式，并且当前 CU 继续划分成四个子 CU。

表 4.1 H.264 intra16x16 块在 HEVC 中的分布

	Depth 0	Depth 1	Depth 2	Depth 3
352x288	1.55%	26.22%	51.71%	20.53%
416x240	0.16%	10.41%	52.39%	37.05%
832x480	4.79%	34.56%	53.14%	7.52%
1280x720	12.77%	35.94%	37.02%	14.27%
Average	4.82%	26.78%	48.56%	19.84%

表 4.2 H.264 intra8x8 块在 HEVC 中的分布

	Depth 0	Depth 1	Depth 2	Depth 3
CIF	0.71%	11.25%	22.57%	65.48%
416x240	0.14%	6.35%	28.52%	64.98%
832x480	0.89%	14.00%	33.09%	52.02%
720P	3.41%	14.12%	27.20%	55.27%
Average	1.29%	11.43%	27.84%	59.44%

表 4.3 H.264 intra4x4 块在 HEVC 中的分布

	Depth 0	Depth 1	Depth 2	Depth 3
CIF	0.03%	1.43%	6.64%	91.89%
416x240	0.01%	0.41%	4.92%	94.66%
832x480	0.02%	1.58%	9.60%	88.80%
720P	0.41%	2.32%	10.71%	86.55%
Average	0.12%	1.44%	7.97%	90.48%

对于深度大于或等于 2 的 CU，本算法直接利用 H.264 的宏块类型进行映射。表 4.1 为 H.264 的 intra16x16 块在 HEVC 的深度分布的统计表，从表中可以看出 H.264 的 intra16x16 块主要分布在深度为 2 的 CU 中，在深度为 1 和 3 的 CU 也有一定比例的分布。表 4.2 表示了 H.264 的 intra8x8 块在 HEVC 的深度分布，从表中可以看出 H.264 的 intra8x8 块主要分布在深度为 3 的 CU 中，在深度为 2 的 CU 中也有一定比例的分布，在深度小于 2 的 CU 中，分布的概率比较小。表 4.3 表示了 H.264 中的 intra4x4 块在 HEVC 的深度分布，从表中可以发现，H.264 的 intra4x4 块主要分布在深度为 3 的 CU 中，在其它

深度分布的概率均很小。基于表 4.1 至 4.3 的统计特性，对于深度等于 2 的 CU，我们遵循如下的转码算法：

1. 如果当前 CU 对应的 H.264 的宏块类型为 intra16x16 块，进行当前深度的 PART_2Nx2N 模式，并且当前 CU 进一步划分。
2. 如果当前 CU 对应的 H.264 宏块类型为 intra8x8 块，当前 CU 直接划分。
3. 如果当前 CU 对应的 H.264 宏块类型为 intra4x4 块，当前 CU 直接划分。

对于深度等于 3 的 CU，我们遵循如下的转码算法：

1. 如果当前 CU 所对应的 H.264 的宏块类型为 intra16x16 块，进行当前深度的 PART_2Nx2N 模式。
2. 如果当前 CU 所对应的 H.264 宏块的类型为 intra8x8，进行当前深度的 PART_2Nx2N 模式及 PART_NxN 模式。
3. 如果当前 CU 所对应的 H.264 宏块类型为 intra4x4，进行当前深度的 PART_2Nx2N 模式及 PART_NxN 模式。

4.3.2 帧间快速转码算法

由于帧间转码中存在 H.264 的运动矢量信息，因此，除 DCT 非零系数个数及 DCT 系数能量这两个特征值外，在帧间快速转码中，还可使用运动矢量方差距离作为特征值，其计算公式如式（4.6）所示。

$$\nu = \sqrt{(\sigma_x^{\,2})^2 + (\sigma_y^{\,2})^2} \tag{4.6}$$

$$MV_{scale} = \frac{POC_{cur} - POC_{scale}}{POC_{cur} - POC_{ref}} MV_{ori} \tag{4.7}$$

其中，$\sigma_x^{\,2}$和$\sigma_y^{\,2}$分别表示当前 CU 所包含的所有 H.264 运动矢量在水平分量及垂直分量上的方差。在计算当前 CU 的运动矢量方差距离前，所有的 H.264 运动矢量信息必须先利用式（4.7）缩放至同一参考帧，否则由参考帧不同的运动矢量计算所得的特征值毫无意义。

在利用 4.2 节介绍的阈值计算方法获取帧间阈值后，我们便可利用当前 CU 的特征值与阈值之间的关系进行帧间的快速转码。与帧内快速转码算法类似，对于深度小于 2 的 CU，我们采用固定阈值下的快速转码算法，对于深度大于或等于 2 的 CU，我们利用 H.264 的宏块类型与 CU 划分的统计信息直接

进行映射。对于深度小于 2 的 CU，其具体转码算法为：

1. 如果当前 CU 的特征值 f <= Tlow，只进行当前深度的 Merge 模式与 PART_2Nx2N 模式，并且当前 CU 进一步划分；
2. 如果当前 CU 的特征值 f >= Thigh，直接将当前 CU 划分成四个子 CU；
3. 如果当前 CU 的特征值 f 介于两者之间，当前 CU 下的所有 inter 模式均进行，并且当前 CU 进一步划分；
4. 如果当前 CU 的特征值 f 不可得，即当前 CU 包含 H.264 的 intra 块，则当前 CU 下的所有 inter 及 intra 模式均进行，并且当前 CU 进一步划分。

表 4.4 H.264 各宏块类型在深度为 2 的 CU 中的概率分布

	Skip/direct	16x16	16x8	8x16	8x8	Intra
Merge	**96.17%**	**32.00%**	**17.79%**	**17.34%**	**5.35%**	**18.78%**
2Nx2N	2.20%	**36.63%**	**19.03%**	**18.58%**	**9.93%**	**18.58%**
2NxN	0.21%	1.76%	**14.19%**	1.64%	**3.20%**	4.64%
Nx2N	0.34%	2.52%	2.00%	**19.19%**	**9.06%**	0.77%
NxN	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
2NxnU	0.02%	1.14%	2.08%	0.95%	1.91%	0.34%
2NxnD	0.02%	0.86%	1.72%	0.78%	1.65%	0.25%
nLx2N	0.06%	1.30%	1.10%	2.83%	2.93%	0.45%
nRx2N	0.06%	1.04%	0.86%	2.20%	2.50%	0.39%
Intra	0.10%	1.81%	2.19%	2.36%	0.87%	**35.26%**
Split	0.84%	**21.94%**	**39.03%**	**34.13%**	**62.58%**	**20.54%**

表 4.4 和表 4.5 分别显示了 H.264 各个宏块类型在深度为 2 和深度为 3 的 CU 中的 PU 模式的概率分布，其中，每一列代表一种 H.264 宏块类型的分布。从表 4.4 可以看出，对于 skip/direct 宏块，它所对应的 CU 有 96%的概率选择 Merge 模式，因此，在进行帧间转码时，若当前深度为 2 的 CU 所对应的 H.264 宏块类型恰为 skip/direct，那么当前 CU 的 PU 模式直接选为 Merge 模式即可，并且当前 CU 不再继续往下划分。对于 16x16 宏块，它所对应的 CU 分别有 32%、36%及 22%的概率选择 Merge 模式、PART_2Nx2N 模式及 split 模式，因此对于深度为 2 的 CU，若其对应的 H.264 宏块类型恰为 16x16，则当前 CU 的 PU 模式只需进行 Merge 及 PART_2Nx2N 的 RDO 决策，并且当前 CU 需进一步划分成四个子 CU。其它 CU 的划分决策及 PU 选择可依此类推。基

于此表 4.4 及表 4.5 的统计信息，我们设计了 H.264 的宏块类型与当前 CU 的划分决策及 PU 选择的映射表格，详见表 4.6 及表 4.7。

表 4.5 H.264 各宏块类型在深度为 3 的 CU 中的概率分布

	Skip	16x16	16x8	8x16	8x8	8x4	4x8	4x4	Intra
Merge	**79.19%**	**51.34%**	**46.09%**	**45.58%**	**40.61%**	**23.23%**	**22.53%**	**6.62%**	**12.60%**
2Nx2N	**15.03%**	**41.29%**	**45.56%**	**44.88%**	**49.89%**	**34.11%**	**32.12%**	**33.84%**	**21.00%**
2NxN	1.19%	3.17%	4.31%	3.98%	4.30%	**18.55%**	6.46%	**29.83%**	2.34%
Nx2N	1.59%	4.16%	3.76%	5.33%	4.97%	8.99%	**23.76%**	**28.82%**	3.52%
NxN	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
2NxnU	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
2NxnD	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
nLx2N	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
nRx2N	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Intra	3.00%	0.03%	0.28%	0.23%	0.23%	**15.12%**	**15.14%**	0.89%	**60.54%**
Split	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

表 4.6 CU 深度为 2 下的 H.264 宏块类型与 CU、PU 映射关系表

	Skip/direct	16x16	16x8	8x16	8x8	Intra
Merge	**X**	**X**	**X**	**X**	**X**	**X**
2Nx2N		**X**	**X**	**X**	**X**	**X**
2NxN			**X**		**X**	
Nx2N				**X**	**X**	
NxN						
2NxnU						
2NxnD						
nLx2N						
nRx2N						
Intra						**X**
Split		**X**	**X**	**X**	**X**	**X**

表 4.7 CU 深度为 3 下的 H.264 宏块类型与 CU、PU 映射关系表

	Skip	16x16	16x8	8x16	8x8	8x4	4x8	4x4	Intra
Merge	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
2Nx2	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
2NxN						**X**		**X**	
Nx2N							**X**	**X**	
NxN									
2Nxn									
2Nxn									
nLx2									
nRx2									
Intra						**X**	**X**		**X**
Split									

4.4 转码框架

图 4.2 为固定阈值下的快速转码算法的流程图，它分成两部分：阈值训练与快速转码。在阈值训练部分，利用 4.2 节介绍的方法进行帧内及帧间的阈值训练。在快速转码部分，利用 4.3 节介绍的帧内及帧间的快速转码算法进行相应的快速转码。图 4.3 进行一步展示了转码阶段的转码框架，与基于 AMV-RDO 的快速转码算法框架不同之处在于，固定阈值下的快速转码算法需要利用离线训练出的阈值与解码信息一起作用于编码控制单元。

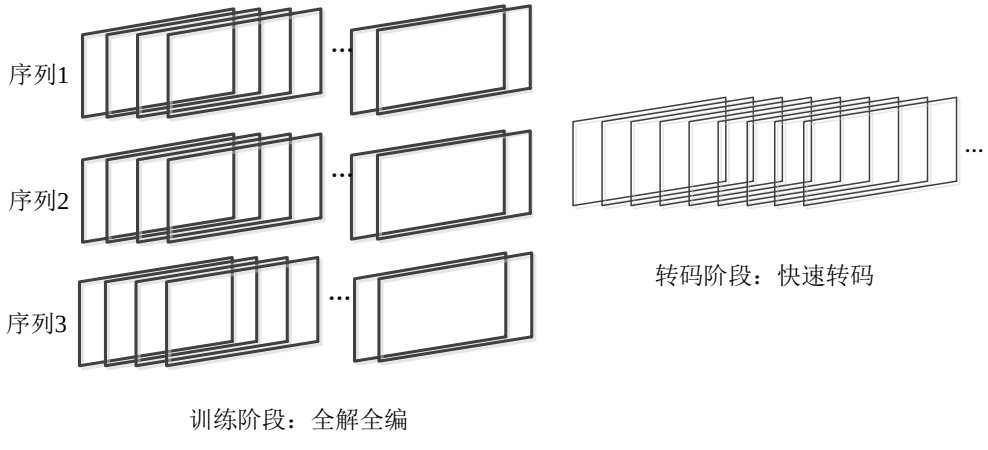


图 4.2 固定阈值训练示意图

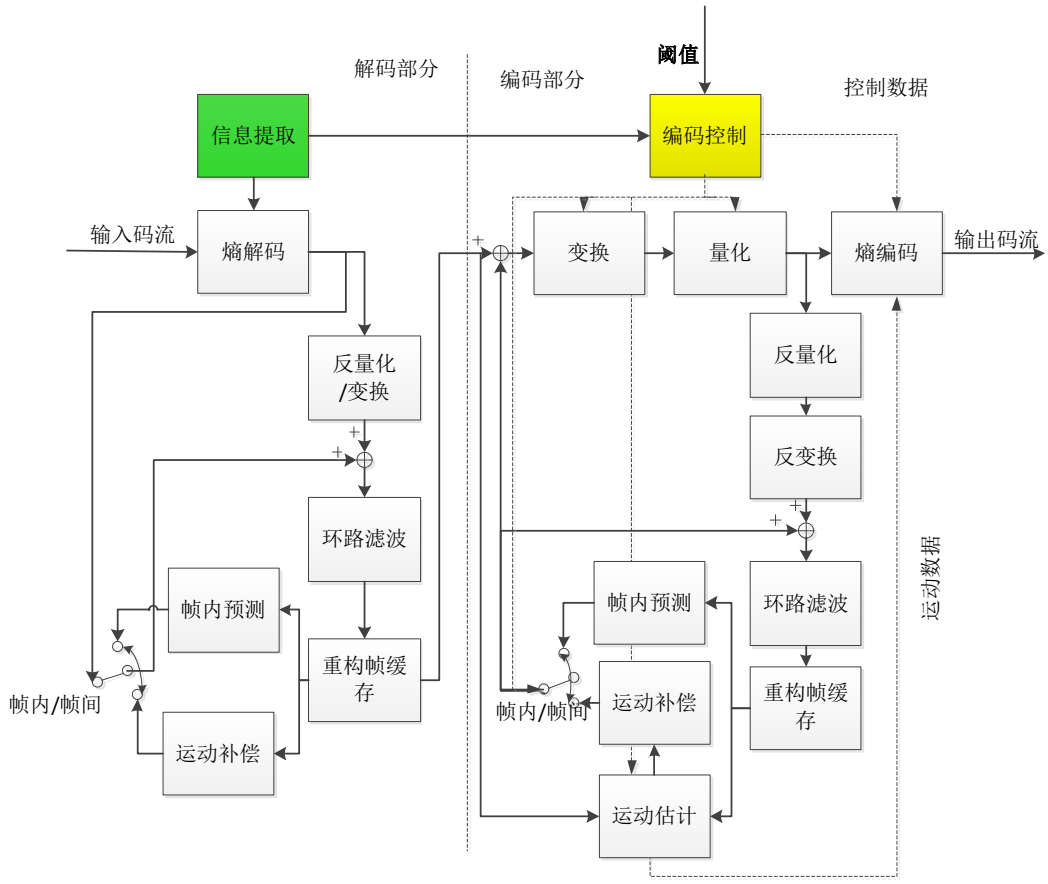


图 4.3 固定阈值下的快速转码算法框架图

4.5 实验结果

4.5.1 帧内快速转码算法性能

利用 4.2 节介绍的阈值计算方法得到的帧内阈值详见表 4.8，其中，Coeff_Num_Y 代表 Y 分量的 DCT 非零系数个数，Coeff_Num_Total 代表 Y、U、V 分量的 DCT 非零系数个数总和，Coeff_Energy_Y 代表 Y 分量的 DCT 系数能量，Coeff_Energy_Total 代表 Y、U、V 三个分量的 DCT 系数能量总和。在进行帧内快速转码算法性能测试时，我们分别以 Coeff_Num_Y、Coeff_Num_Total、Coeff_Energy_Y、Coeff_Energy_Total 作为当前 CU 的特征，转码的 H.264 序列采用全 I 帧 H.264 码流，共测试 CIF、416x240、832x480 及 720P 分辨率下的 14 个序列。

表 4.8 各个特征下的帧内阈值

	Coeff_Num_Y	Coeff_Num_Total	Coeff_Energy_Y	Coeff_Energy_Total
Thigh	49	66	579436	868816
Tlow	2	2	5408	17875

表 4.9 以 DCT 非零系数个数为特征值时的帧内快速转码算法性能

分辨率	序列	Coeff_Num_Total		Coeff_Num_Y	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	4.14%	1.83	4.14%	1.83
	container	3.61%	1.84	3.60%	1.83
	flower	1.84%	1.86	1.85%	1.86
416x240	BasketballPass	4.43%	1.89	4.43%	1.89
	BQSquare	1.07%	1.98	1.07%	1.98
	RaceHorses	2.18%	1.89	2.17%	1.89
832x480	BasketballDrill	2.31%	2.08	2.32%	2.08
	BQMall	2.49%	2.00	2.51%	2.00
	RaceHorses	2.78%	1.95	2.75%	1.95
720P(1280x720)	FourPeople	3.46%	1.85	3.57%	1.85
	Johnny	9.04%	1.73	9.13%	1.81
	KristenAndSara	5.77%	1.59	5.88%	1.71
	Vidyo1	6.57%	1.71	6.85%	1.73
	Vidyo3	8.14%	1.89	8.26%	1.89
CIF(352x288)		3.20%	1.84	3.20%	1.84
416x240		2.56%	1.92	2.56%	1.92
832x480		2.53%	2.01	2.53%	2.01
720P(1280x720)		6.60%	1.75	6.74%	1.80
整体		4.13%	1.86	4.18%	1.88

表 4.9、表 4.10 是分别以 Coeff_Num_Y、Coeff_Num_Total、Coeff_Energy_Y 及 Coeff_Energy_Total 为特征值的快速转码算法的 RD 性能及加速比性能。从表 4.9 与表 4.10 可以看出，四个特征值下的帧内快速转码较全解全编的转码在几乎不降低转码质量的同时，可以提升近一倍的转码速度。我们进一步将不同特征下的 RD 性能及速度性能总结至表 4.11 及表 4.12 中，以分析不同特征值下的转码 RD 性能与速度性能。

表 4.10 以 DCT 系数能量为特征值时的帧内快速转码算法性能

分辨率	序列	Coeff_Energy_Total		Coeff_Energy_Y	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	4.14%	1.83	4.14%	1.82
	container	3.31%	1.81	3.51%	1.83
	flower	1.83%	1.86	1.83%	1.86
416x240	BasketballPass	4.05%	1.87	4.18%	1.88
	BQSquare	1.05%	1.96	1.06%	1.98
	RaceHorses	2.13%	1.89	2.12%	1.89
832x480	BasketballDrill	2.31%	2.08	2.32%	2.08
	BQMall	2.49%	1.96	2.50%	2.00
	RaceHorses	2.63%	1.91	2.61%	1.91
720P(1280x720)	FourPeople	3.23%	1.77	3.36%	1.85
	Johnny	8.75%	1.67	8.86%	1.67
	KristenAndSara	5.29%	1.50	5.35%	1.55
	Vidyo1	5.31%	1.60	5.65%	1.60
	Vidyo3	7.74%	1.79	7.95%	1.84
CIF(352x288)		3.09%	1.83	3.16%	1.84
416x240		2.41%	1.91	2.45%	1.92
832x480		2.48%	1.98	2.48%	2.00
720P(1280x720)		6.06%	1.67	6.23%	1.70
整体		3.88%	1.82	3.96%	1.84

从表 4.11 中可以总结到：对于同一特征，只选其中 Y 分量作为特征值和同时选择 Y、U、V 三个分量共同作为特征值的 RD 性能差异甚微；对于不同特征，DCT 非零系数的个数的 RD 性能略低于 DCT 系数能量的 RD 性能，这是因为，DCT 系数能量需要考虑到每个 DCT 系数的实际大小，更能反映一个 CU 的属性特征，因此，它的 RD 性能略优。在速度性能对比上，从表 4.12 中可以分析得到：对于同一特征，只利用其 Y 分量作为特征值的速度性能略优于以 Y、U、V 三个分量共同作为特征值的速度性能，这是因为同时计算三个分量的运算量要大于单独计算 Y 分量的运算量；对于不同的特征，DCT 系数能量的速度性能要略低于 DCT 非零系数个数的速度性能，这是因为，在 DCT 系数能量的计算过程中需要进行乘法运算，而在 DCT 非零系数个数的计算过程中只需要进行简单的加法运算即可，因此，DCT 系数能量的计算复杂度略高于 DCT 非零系数个数的计算复杂度。

表 4.11 四种特征下的帧内快速转码算法 RD 性能对比

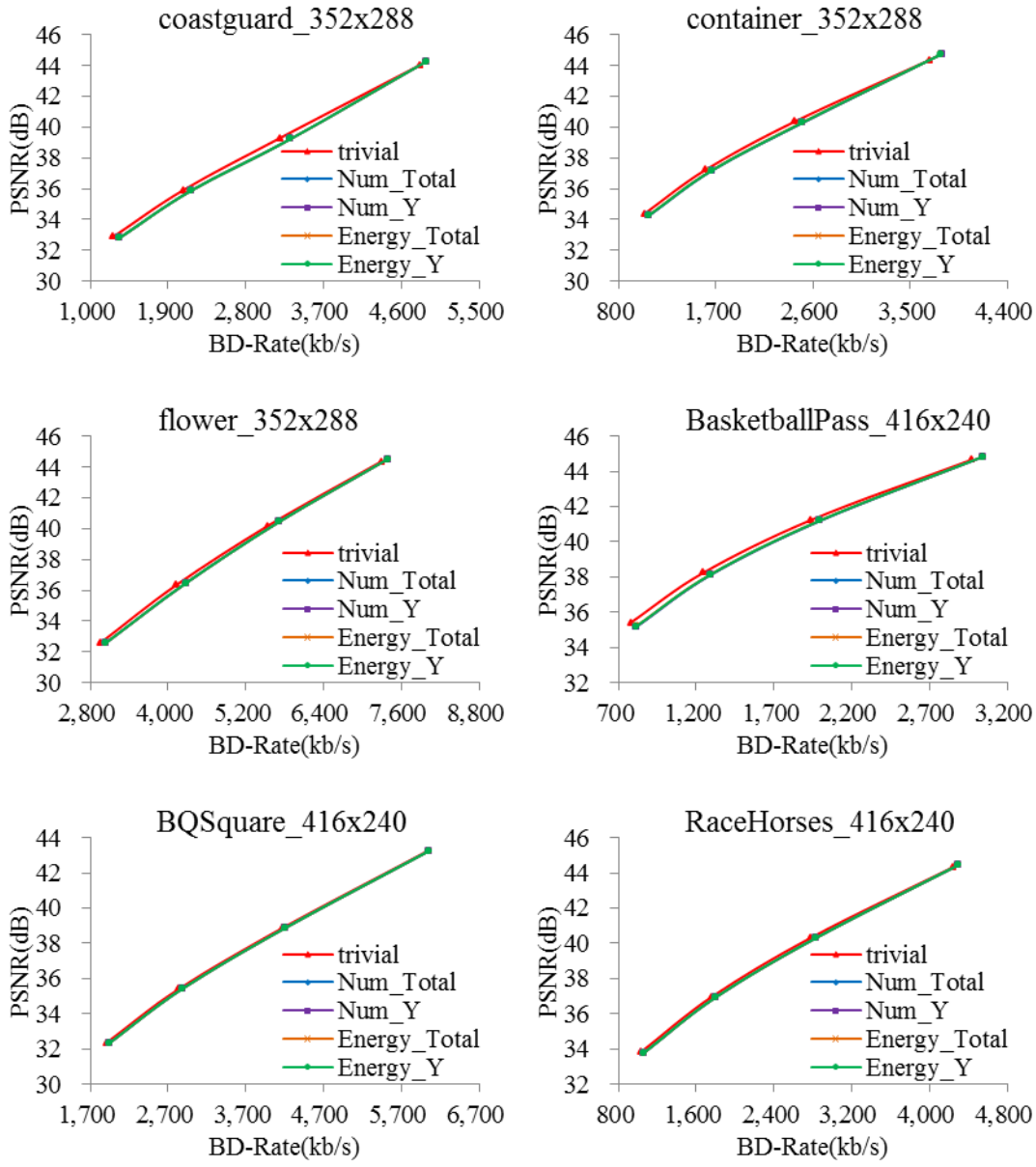
	Coeff_Num_Y	Coeff_Num_Total	Coeff_Energy_Y	Coeff_Energy_Total
CIF(352x288)	3.20%	3.20%	3.16%	3.09%
416x240	2.56%	2.56%	2.45%	2.41%
832x480	2.53%	2.53%	2.48%	2.48%
720P(1080x720)	6.74%	6.60%	6.23%	6.06%
Total	4.18%	4.13%	3.96%	3.88%

表 4.12 四种特征下的帧内快速转码算法速度对比

	Coeff_Num_Y	Coeff_Num_Total	Coeff_Energy_Y	Coeff_Energy_Total
CIF(352x288)	1.84	1.84	1.84	1.83
416x240	1.92	1.92	1.92	1.91
832x480	2.01	2.01	2.00	1.98
720P(1080x720)	1.80	1.75	1.70	1.67
Total	1.88	1.86	1.84	1.82

图 4.4 进一步展示了不同特征下的帧内转码算法的 RD 曲线，其中红线代表全解全编的转码，蓝线代表以 Coeff_Num_Total 为特征的帧内快速转码，紫线代表以 Coeff_Num_Y 为特征的帧内快速转码，橙线代表以

Coeff_Energy_Total 为特征的帧内快速转码，绿线代表以 Coeff_Energy_Y 为特征的帧内快速转码。从图 4.4 可以看出，四种特征下的快速转码 RD 曲线十分接近于全解全编转码的 RD 曲线，且四种特征下的 RD 曲线十分接近，几近重合。因此，图 4.4 也近一步表明，四种特征值下的帧内快速转码算法的 RD 性能较全解全编转码几乎没有损失。



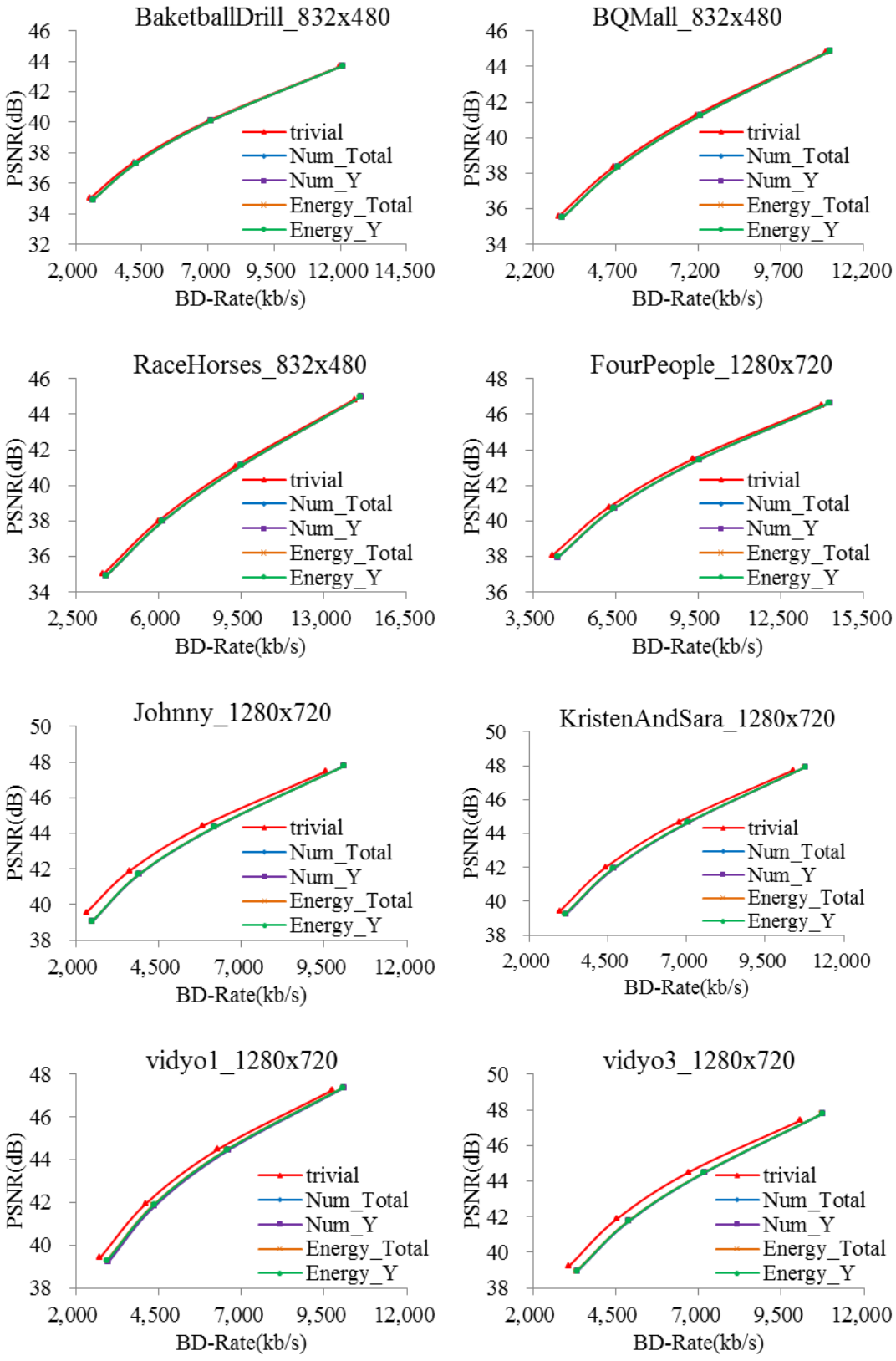


图 4.4 不同特征下的帧内快速转码 RD 曲线对比图

4.5.2 帧间快速转码算法性能

利用 4.2 节介绍的阈值计算方法获取的帧间阈值详见表 4.13，其中，Coeff_Num_Y 代表 DCT 非零系数的个数，Coeff_Energy_Y 代表 DCT 系数能量之和，MV_Distance 代表运动矢量的方差距离。由于在 4.5.1 节中介绍的帧内快速转码算法性能分析中，我们已经得知，对于同一特征只利用其 Y 分量作为特征值得到的转码 RD 性能与同时使用 Y、U、V 三个分量作为特征值得到的转码 RD 性能几乎没有差异，但前者的速度性能略优于后者，因此，在帧间快速转码算法的性能对比中，每个特征的计算只使用其 Y 分量，而不再使用 Y、U、V 三个分量之和。在进行帧间快速转码性能测试时，我们只打开帧间的快速转码算法，对于帧内转码，我们仍旧采用全解全编的转码方式以更客观的评价帧间快速转码算法的性能。测试的 H.264 码流结构为 IPPP 结构，共测试了 CIF、416x240、832x480 及 720P 分辨率下的 14 个序列。

表 4.13 各个特征下的帧间阈值

	Coeff_Num_Y	Coeff_Energy_Y	MV_Distance
Thigh	59	952916	1237
Tlow	0	0	0

表 4.14 显示了三个特征下的转码 RD 性能与速度性能。从表中可以发现：三个特征值下的 RD 性能较全解全编的转码几乎没有损失，但转码加速较全解全编转码提升了至少一倍；对比三个特征值，运动矢量方差距离的 RD 性能最优、其次是 DCT 系数能量，最后是 DCT 非零系数个数；在速度性能上，则刚好相反，DCT 非零系数个数的速度最优，较全解全编转码的加速近两倍，其次是 DCT 系数能量，最后是运动矢量方差距离。DCT 系数能量的 RD 性能优于 DCT 非零系数个数的 RD 性能的原因已在 4.5.1 中进行了阐述，而运动矢量方差距离的 RD 性能优于 DCT 系数能量及 DCT 非零系数个数则是因为，对于 H.264 编码的运动矢量与 HEVC 编码的运动矢量的大体方向应该接近，使用 H.264 的运动矢量的方差距离可以更直观的反映当前 CU 块的整体运动趋势，从而可以较优的决策当前 CU 是否需要进一步的划分。但由于运动矢量方差距离需要先对运动矢量进行缩放，并分别求解水平分量与垂直分量上的方差值，最后，还需利用水平方差与垂直方差进行最终运动矢量方差距离

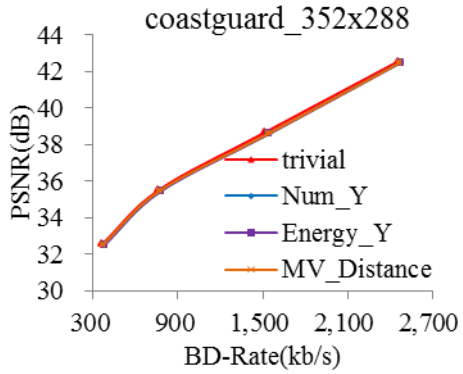
的求解，因此，在求解过程中，会带来大量的乘法、除法甚至开方运算，因此，它的计算复杂度也最高，从而导致它的转码速度性能最低。

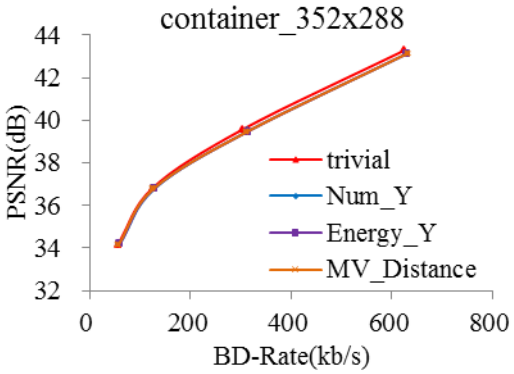
表 4.14 不同特征值下的帧间快速转码算法 RD 性能与速度性能

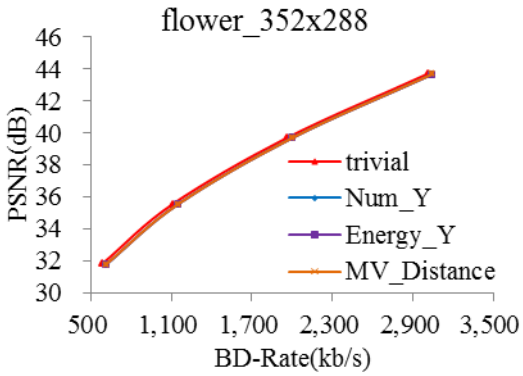
分辨率	序列	Coeff_Num_Y		Coeff_Energy_Y		MV_Distance	
		BD-Rate	加速	BD-Rate	加速	BD-Rate	加速
CIF (352x288)	coastguard	3.17%	3.89	3.06%	3.69	2.25%	1.82
	container	4.74%	2.66	4.58%	2.48	3.96%	1.80
	flower	3.01%	3.47	3.01%	3.56	2.95%	1.69
416x240	BasketballPass	4.03%	2.24	4.03%	2.24	3.99%	1.57
	BQSquare	4.48%	3.81	4.46%	3.91	4.15%	1.78
	RaceHorses	4.31%	2.41	4.31%	2.43	4.22%	1.51
832x480	BasketballDrill	5.52%	2.18	5.49%	2.18	5.27%	1.80
	BQMall	4.54%	2.73	4.47%	2.73	3.93%	1.73
	RaceHorses	3.67%	2.46	3.61%	2.46	3.52%	1.79
720P (1280x720)	FourPeople	4.42%	2.26	4.10%	2.13	3.51%	1.76
	Johnny	6.77%	2.21	5.55%	2.07	3.47%	1.67
	KristenAndSara	4.65%	1.95	4.10%	1.69	3.07%	1.55
	Vidyo1	4.31%	2.38	4.43%	2.38	3.11%	1.67
	Vidyo3	5.65%	2.38	5.84%	2.50	4.91%	1.67
CIF(352x288)		3.64%	3.34	3.55%	3.24	3.05%	1.77
416x240		4.27%	2.82	4.27%	2.86	4.12%	1.62
832x480		4.58%	2.46	4.52%	2.46	4.24%	1.77
720P(1280x720)		5.16%	2.24	4.80%	2.15	3.61%	1.66
整体		4.52%	2.65	4.36%	2.60	3.74%	1.70

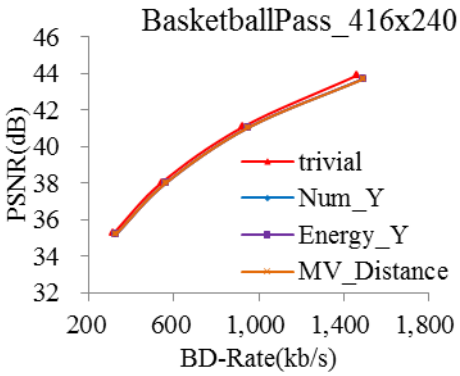
图 4.5 进一步展示了不同特征下的帧间转码的 RD 曲线对比情况，其中红线代表全解全编转码的 RD 曲线，蓝线代表以 Coeff_Num_Y 为特征的帧间快速转码的 RD 曲线，紫线代表以 Coeff_Energy_Y 为特征的帧间快速转码的 RD 曲线，橙线代表以 MV_Distance 为特征的帧间快速转码的 RD 曲线。从图 4.5

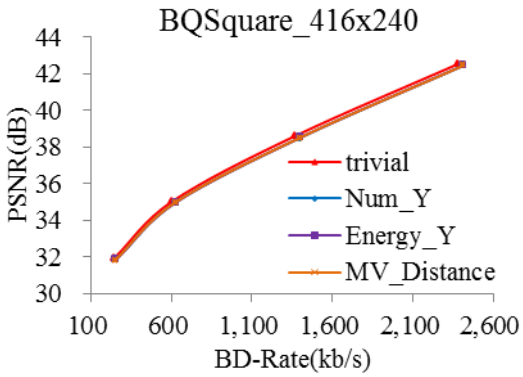
中可以看出，三条快速转码算法的 RD 曲线均十分接近于全解全编转码的 RD 曲线，但以橙色曲线最为接近。

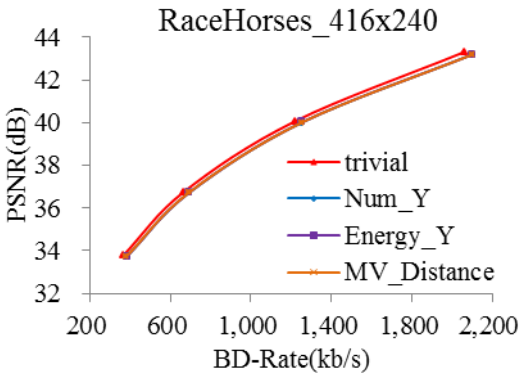












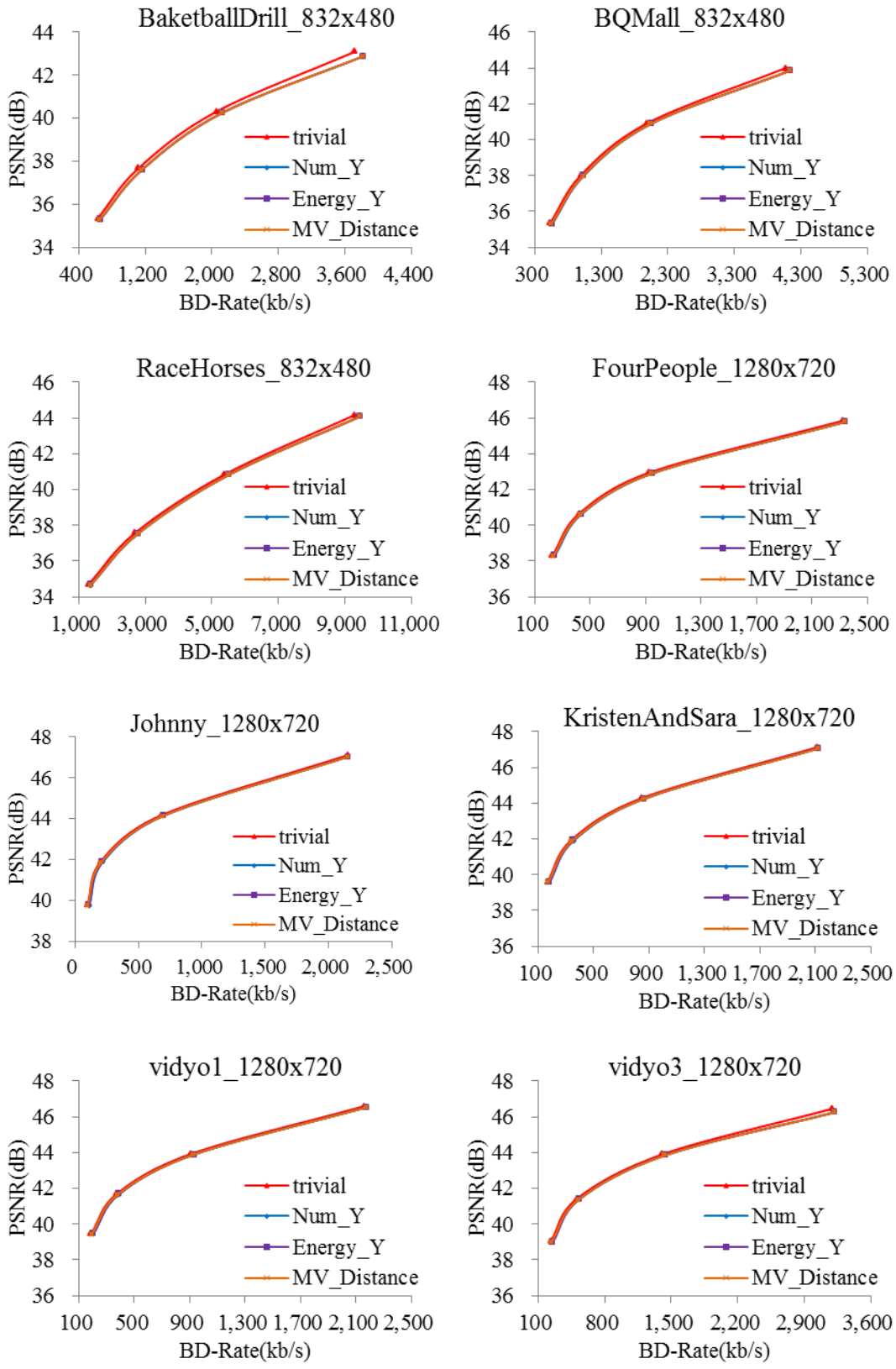


图 4.5 不同特征下的帧间快速转码 RD 曲线对比图

4.5.3 整体算法性能

表 4.15 固定阈值下的整体算法性能与文献[66]算法性能对比

分辨率	序列	固定阈值快速转码		文献[66]	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	4.12%	2.52	4.20%	1.67
	container	4.21%	2.54	4.73%	2.06
	flower	3.48%	3.40	3.46%	1.93
416x240	BasketballPas	4.36%	2.16	5.04%	1.70
	BQSquare	4.99%	4.00	4.99%	2.28
	RaceHorses	5.11%	2.23	4.75%	1.69
832x480	BasketballDril	4.84%	2.15	5.13%	1.82
	BQMall	4.69%	2.32	5.16%	1.64
	RaceHorses	4.07%	1.73	3.80%	1.83
720P(1280x720)	FourPeople	4.39%	2.02	4.75%	1.62
	Johnny	5.55%	1.89	3.71%	1.50
	KristenAndSa	4.52%	1.76	4.17%	1.42
	Vidyo1	5.35%	2.01	4.97%	1.54
	Vidyo3	6.09%	2.25	5.07%	1.85
CIF		3.94%	2.82	4.13%	1.89
416x240		4.82%	2.80	4.93%	1.89
832x480		4.53%	2.07	4.70%	1.76
720P		5.18%	1.99	4.53%	1.59
整体		4.70%	2.36	4.57%	1.75

除了单独进行帧内快速转码算法与帧间快速转码算法的性能测试外，我们还进行了整体的快速转码算法性能测试。测试 H.264 码流结构为 IBBP，共测试 CIF、416x240、832x480 及 720P 分辨率下的 14 个序列。由于 Coeff_Energy_Y 特征值在 RD 性能与速度性能上有较好的折衷，因此，帧内与帧间的转码均使用 Coeff_Energy_Y 作为特征值。B、P 帧使用相同的阈值，具体阈值可见表 4.13。为了更好的说明本章提出的固定阈值下的快速转码算法的有效性，我们进一步在目前全解全编转码框架中实现了文献[66]提出的算法以进行性能对比。固定阈值的快速转码算法与文献[66]的算法性能总结至表

4.15，从表中可以看出，本算法在提供相同转码质量的同时，转码速度比算法[66]的速度更快。对于序列 CIF 分辨率下的 flower 序列、416x240 分辨率下的 BQSquare 序列，加速比甚至超过 1 倍，平均情况下速度提升了 0.61 倍。相比于全解全编的转码算法，本章提出的算法在几乎不损失 RD 性能的同时，转码速度提升 1.36 倍，即节省了 58%的转码时间。

4.6 本章小结

本章提出了一种固定阈值下的快速转码算法，它利用当前 CU 的特征值与阈值之间的关系进行当前 CU 的划分决策与 PU 选择。实验结果表明，本算法较全解全编转码在几乎不带来性能损失的同时，转码速度提升 1.36 倍。实验中，我们还进一步的对比了不同特征值下的 RD 性能与速度性能。对于同一个特征值，只使用其中的 Y 分量计算特征值与同时使用 Y、U、V 三个分量计算特征值的 RD 性能几乎没有差异，但只使用 Y 分量计算特征值的速度性能略优于同时使用三个分量计算特征值。对于不同特征值之间的性能对比，实验结果表明，运动矢量方差距离具有最优的 RD 性能，但转码速度最慢，DCT 系数能量的 RD 性能与转码速度均处于中间地位，而 DCT 非零系数个数的 RD 性能则最差，但转码速度最优。

第 5 章 阈值自适应的快速转码算法

5.1 引言

固定阈值下的快速转码算法具有良好的 RD 性能及转码速率，然而，该算法下的阈值经离线训练获取，在快速转码中所有序列均使用相同的阈值，固定阈值是否适用于特征各不相同的序列是一个值得进一步探究的问题。为此，本章介绍一种阈值自适应的快速转码算法，在本算法中，阈值将根据序列特性、编码 QP、帧类型进行自适应调整。实验结果表明，相比固定阈值的快速转码，本算法具有更好的 RD 性能，然而，由于阈值自适应的过程是在线训练的过程，因此，本算法的速度性能略低于固定阈值下的快速转码算法。除阈值自适应算法外，本章就目前的快速转码算法还提出一种 PU 快速模式决策算法以进一步提升转码速率。

下面将分别介绍阈值自适应的快速转码算法与 PU 快速模式决策算法，之后，两种算法的实验结果将分别被展示。

5.2 阈值自适应的快速转码算法

5.2.1 算法框架

阈值自适应的快速转码算法的转码框架如图 5.1 所示。转码过程共分成三个部分，第一部分为训练部分，它采用全解全编的转码方式转码序列的前 k 帧，并将 CU 的特征值及 CU 的划分决策进行记录。其中 k 值不固定，它可以作为控制转码算法复杂度的参数。CU 的划分决策记录在数组 C_d 中，CU 的特征值记录在数组 F_d 中，d 代表深度，即不同深度的 CU 的划分决策与特征值将分别记录。第二部分为建模部分，它利用记录的 C_d 与 F_d 进行阈值 Tlow 与 Thigh 的计算，计算方式与固定阈值下的特征值计算方式相同，即对于 Tlow 需满足以下三点要求：

1. 特征值 f<=Tlow 的所有 CU 中，c_i^d 值为 1 的比率达到 90%；
2. 任何小于 Tlow 的特征值作为 Tlow 也满足条件 1；
3. 同时满足条件 1 与条件 2 的所有特征值中的最大值被选为 Tlow；

同理，Thigh 遵循以下三点要求：

1. 特征值 f>=Thigh 的所有 CU 中，c_I^d 值为 0 的比率达到 90%；
2. 任何大于 Thigh 的值作为 Thigh 也满足条件 1；
3. 同时满足条件 1 与条件 2 的所有特征值中的最大值被选为 Thigh；

第三部分为快速转码部分，CU 将根据其深度进行不同的快速转码过程。对于深度小于 2 的 CU，通过比较当前 CU 的特征值 f 与阈值 Tlow 及 Thigh 之间的关系进行当前 CU 的划分决策及 PU 模式决策。具体为：

1. 如果当前 CU 的特征值 f <= Tlow，只进行当前深度的 Merge 模式与 PART_2Nx2N 模式，并且当前 CU 进一步划分；
2. 如果当前 CU 的特征值 f >= Thigh，直接将当前 CU 划分成四个子 CU；
3. 如果当前 CU 的特征值 f 介于两者之间，当前 CU 下的所有 inter 模式均进行，并且当前 CU 进一步划分；
4. 如果当前 CU 的特征值 f 不可得，即当前 CU 包含 H.264 的 intra 块，则当前 CU 下的所有 inter 及 intra 模式均进行，并且当前 CU 进一步划分。

对于深度大于或等于 2 的 CU，直接利用当前 CU 所对应的 H.264 宏块类型进行映射，映射表详见表 5.1 与表 5.2。

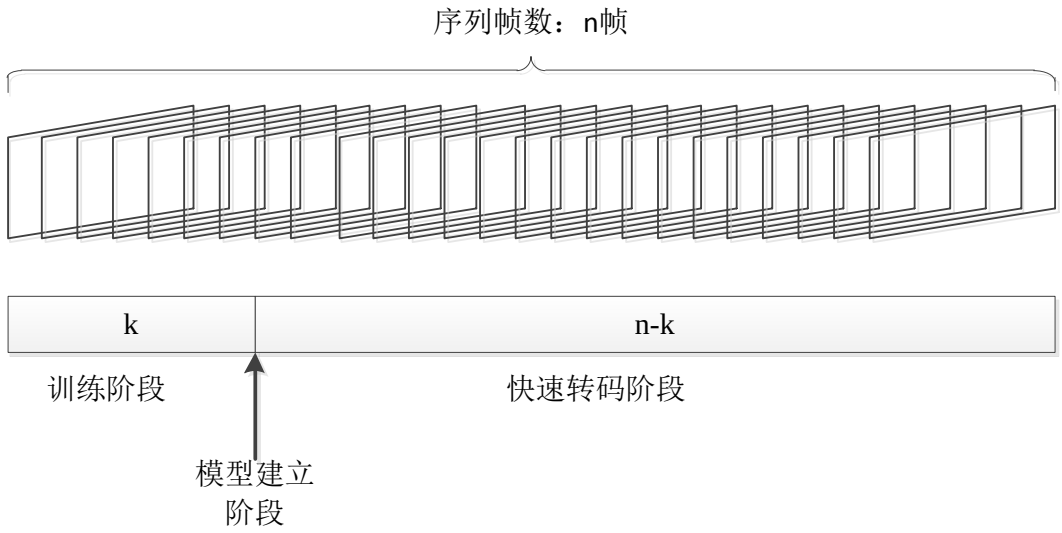


图 5.1 阈值自适应快速转码算法框架

表 5.1 CU 深度为 2 下的 H.264 宏块类型与 CU、PU 映射关系表

	Skip/direct	16x16	16x8	8x16	8x8	Intra
Merge	**X**	**X**	**X**	**X**	**X**	**X**
2Nx2N		**X**	**X**	**X**	**X**	**X**
2NxN			**X**		**X**	
Nx2N				**X**	**X**	
NxN						
2NxnU						
2NxnD						
nLx2N						
nRx2N						
Intra						**X**
Split		**X**	**X**	**X**	**X**	**X**

表 5.2 CU 深度为 3 下的 H.264 宏块类型与 CU、PU 映射关系表

	Skip	16x16	16x8	8x16	8x8	8x4	4x8	4x4	Intra
Merge	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
2Nx2	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
2NxN						**X**		**X**	
Nx2N							**X**	**X**	
NxN									
2Nxn									
2Nxn									
nLx2									
nRx2									
Intra						**X**	**X**		**X**
Split									

5. 2. 2 转码流程

图 5.2 为阈值自适应的快速转码算法流程图。对于输入 CTU，它首先判断当前已训练帧数是否超过设定的训练帧数 k，若未超过，则当前 CTU 使用全解全编的转码方式，即编码端需进行完整 RDO 过程，以选择最优的 CU 划分及 PU 模式，并将决策结果及 CU 特征值记录下来。若当前已训练帧数超过

设定的训练帧数 k，则当前 CTU 使用快速转码算法。对于深度小于 2 的 CU，利用当前 CU 的特征值与阈值之间的关系进行快速转码，对于深度大于或等于 2 的 CU，直接利用当前 CU 对应的 H.264 宏块类型信息进行 CU 划分与 PU 决策的映射。

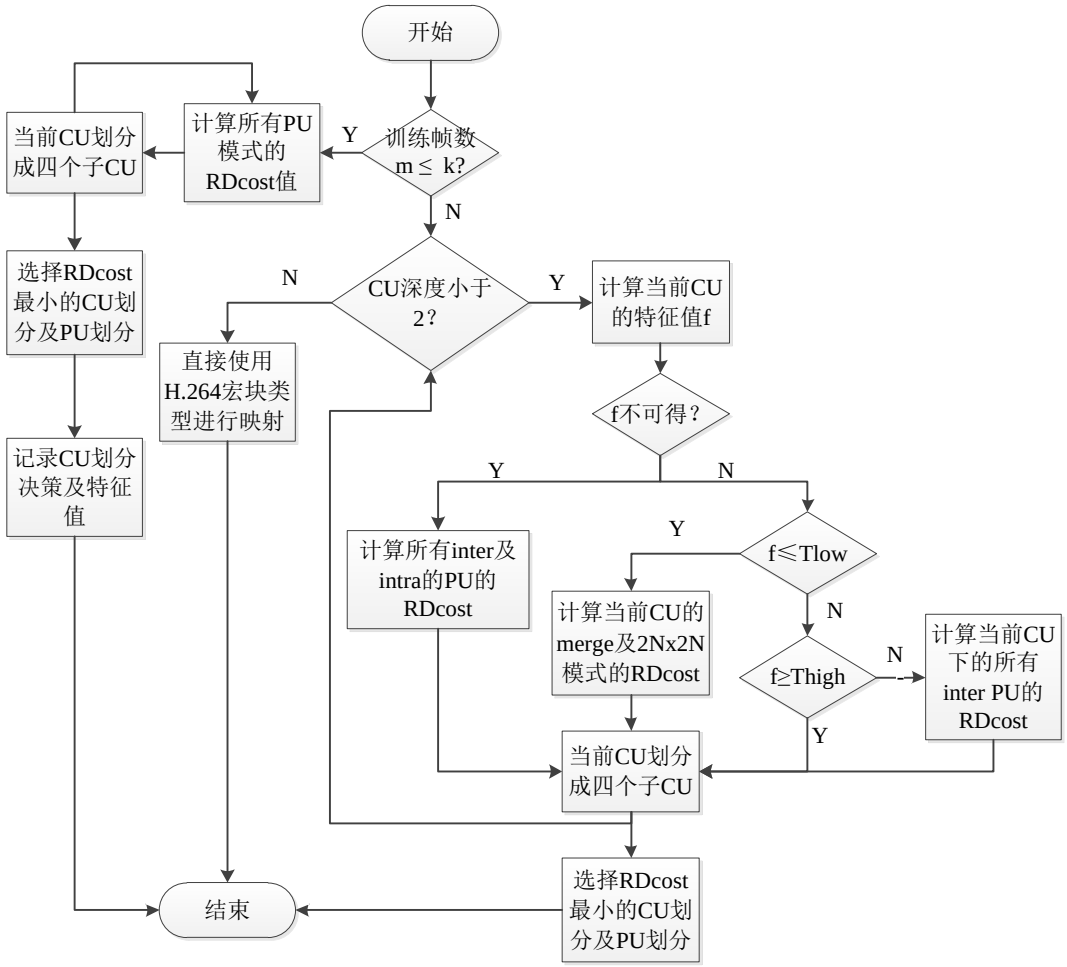


图 5.2 阈值自适应的快速转码算法流程图

5.3 PU 快速模式决策算法

式（5.1）定义了 PU 块运动矢量方差的计算方法，其中 N 表示当前 PU 块所包含的运动矢量数，μ_{MV} 为当前 PU 块包含的运动矢量的平均值。2NxN 模式及 Nx2N 模式的运动矢量方差可通过式（5.2）及式（5.3）求得。其中 $VAR_{2N\times N_{(U)}}$ 及 $VAR_{2N\times N_{(D)}}$ 分别代表 2NxN 模式下上边 PU 块与下边 PU 块的运动矢量方差，通过将上边 PU 块与下边 PU 块的运动矢量方差求平均即可得到 2NxN 模式的运动矢量方差，Nx2N 模式的运动矢量方差同理可得。分别求得

2NxN 及 Nx2N 模式的运动矢量方差后，便可进行这两个模式的跳过决策。当满足式（5.4）时，2NxN 模式被跳过，当满足式（5.5）时，Nx2N 模式被跳过。其中，$\overline{VAR}_{2N\times N\_x}$ 和 $\overline{VAR}_{N\times 2N\_x}$ 分别代表 2NxN 模式及 Nx2N 模式的水平分量的运动矢量方差，$\overline{VAR}_{2N\times N\_y}$ 和 $\overline{VAR}_{N\times 2N\_y}$ 分别代表 2NxN 模式及 Nx2N 模式的垂直分量的运动矢量方差。之所以可利用式（5.4）与式（5.5）进行模式跳过是因为，当一个模式的运动矢量方差较大时，我们认为它包含的 H.264 的运动矢量差异较大，使用当前模式进行预测将不准确，因此，这个模式被选中的概率很小可直接跳过。

$$VAR = \frac{1}{N}\sum_{n=1}^{N}(MV_n - \mu_{MV})^2 \tag{5.1}$$

$$\overline{VAR_{2N\times N}} = \frac{1}{2}\left(VAR_{2N\times N_{(U)}} + VAR_{2N\times N_{(D)}}\right) \tag{5.2}$$

$$\overline{VAR_{N\times 2N}} = \frac{1}{2}\left(VAR_{N\times 2N_{(L)}} + VAR_{N\times 2N_{(R)}}\right) \tag{5.3}$$

$$\overline{VAR}_{2N\times N\_x} \geq 2\times\overline{VAR}_{N\times 2N\_x} \cap \overline{VAR}_{2N\times N\_y} \geq 2\times\overline{VAR}_{N\times 2N\_y} \tag{5.4}$$

$$\overline{VAR}_{N\times 2N\_x} \geq 2\times\overline{VAR}_{2N\times N\_x} \cap \overline{VAR}_{N\times 2N\_y} \geq 2\times\overline{VAR}_{2N\times N\_y} \tag{5.5}$$

对于 AMP 模式的跳过，本算法利用当前 CU 所包含的 H.264 块的残差进行判别。式（5.6）定义了 PU 块的残差计算方式，其中 x_i, y_j 分别代表 H.264 宏块在水平及垂直方向的位置。如果 2NxN 模式下的上边 PU 块的残差大于下边 PU 块的残差，那么 PART_2NxnD 模式被跳过，反之，PART_2NxnU 模式被跳过。如果 Nx2N 模式下的左边 PU 块的残差大于右边 PU 块的残差，那么 nRx2N 模式被跳过，反之，nLx2N 模式被跳过。之所以可以利用残差进行模式跳过是因为，大残差说明当前块所包含的细节多，若利用该模式进行预测将不准确，因此，相应模式被选中的概率很小。

$$residual = \sum_i\sum_j \left|coeff\,(x_i, y_j)\right| \tag{5.6}$$

本节描述的 PU 快速模式决策算法只适用于深度小于 2 的 CU，对于深度大于或等于 2 的 CU，直接利用当前 CU 对应的 H.264 宏块类型映射即可。若将该 PU 快速模式决策算法融入阈值自适应的快速转码算法中，它只适用于特征值介于 Tlow 与 Thigh 之间的 CU 的 PU 决策或特征值不可得的 CU 的 PU 决策，这是因为，在其它情况下，当前 CU 直接被划分或只需进行 Merge 及 PART_2Nx2N 模式的决策，本算法并不适用。

5.4 实验结果

5.4.1 阈值自适应的快速转码算法性能

为检测阈值自适应快速转码算法的有效性，我们分别测试了训练帧数为 5 帧、10 帧、15 帧及 20 帧的阈值自适应的快速转码算法性能，并将其与固定阈值的转码算法性能进行比较。测试 H.264 码流为 CIF、416x240、832x480 及 720P 下的共 11 个序列，码流结构为 IBBP。在测试时，B、P 帧使用相同阈值，但不同深度 CU 训练不同阈值，Coeff_Energy_Y 作为当前 CU 的特征值。

表 5.3 至表 5.6 分别展示了训练帧数为 5、10、15 及 20 帧的阈值自适应的快速转码算法性能。从表 5.3 至表 5.6 可以看出，阈值自适应的快速转码算法比固定阈值的快速转码算法更加有效，各个序列的 RD 性能均有所提升。然而，由于需要在线训练阈值，因此，阈值自适应的快速转码算法的速度性能均低于固定阈值的快速转码算法的速度性能。表 5.7 与图 5.3 进一步总结了各阈值转码算法的平均 RD 性能及速度性能，从表 5.7 及图 5.3 中可以看出，训练帧数越多，快速转码算法的 RD 性能越好，但快速转码算法的速度性能将变差。随着训练帧数增加，转码 RD 性能变好的原因主要有两点：1.随着训练帧数的增加，所选阈值更适用于当前的编码序列；2.训练时进行的全解全编转码具有最优的 RD 性能，因此，随着训练帧数的增加，由全解全编带来的 RD 性能使得整体转码 RD 性能提升。

表 5.3 训练帧数为 5 帧的阈值自适应快速转码算法性能

分辨率	序列	固定阈值		阈值自适应(k=5)	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	4.12%	2.52	3.51%	1.83
	container	4.21%	2.54	3.55%	2.18
	flower	3.48%	3.40	3.18%	2.04
416x240	BasketballPass	4.36%	2.16	4.06%	1.92
	BQSquare	4.99%	4.00	3.73%	2.59
	RaceHorses	5.11%	2.23	4.88%	1.97
832x480	BasketballDrill	4.84%	2.15	4.37%	1.96
	BQMall	4.69%	2.32	4.17%	1.73
	RaceHorses	4.07%	1.73	3.69%	2.04
720P(1280x720)	FourPeople	4.39%	2.02	3.83%	1.65
	Johnny	5.55%	1.89	3.32%	1.54
	KristenAndSara	4.52%	1.76	3.40%	1.42
	Vidyo1	5.35%	2.01	4.00%	1.54
	Vidyo3	6.09%	2.25	4.66%	1.85
CIF(352x288)		3.94%	2.82	3.41%	2.02
416x240		4.82%	2.80	4.22%	2.16
832x480		4.53%	2.07	4.08%	1.91
720P(1280x720)		5.18%	1.99	3.84%	1.60
整体		4.70%	2.36	3.88%	1.88

表 5.4 训练帧数为 10 帧的阈值自适应快速转码算法性能

分辨率	序列	固定阈值		阈值自适应(k=10)	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	4.12%	2.52	3.39%	1.92
	container	4.21%	2.54	3.57%	2.15
	flower	3.48%	3.40	3.11%	2.11
416x240	BasketballPass	4.36%	2.16	4.06%	2.04
	BQSquare	4.99%	4.00	3.67%	2.49
	RaceHorses	5.11%	2.23	4.68%	1.86
832x480	BasketballDrill	4.84%	2.15	4.33%	1.71
	BQMall	4.69%	2.32	4.13%	1.69
	RaceHorses	4.07%	1.73	3.57%	2.04
720P(1280x720)	FourPeople	4.39%	2.02	3.98%	1.69
	Johnny	5.55%	1.89	3.19%	1.57
	KristenAndSara	4.52%	1.76	3.41%	1.49
	Vidyo1	5.35%	2.01	3.89%	1.56
	Vidyo3	6.09%	2.25	4.47%	1.75
CIF(352x288)		3.94%	2.82	3.36%	2.06
416x240		4.82%	2.80	4.14%	2.13
832x480		4.53%	2.07	4.01%	1.81
720P(1280x720)		5.18%	1.99	3.79%	1.61
整体		4.70%	2.36	3.82%	1.86

表 5.5 训练帧数为 15 帧的阈值自适应快速转码算法性能

分辨率	序列	固定阈值		阈值自适应(k=15)	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	4.12%	2.52	3.45%	1.67
	container	4.21%	2.54	3.40%	1.85
	flower	3.48%	3.40	3.13%	1.86
416x240	BasketballPass	4.36%	2.16	4.11%	1.81
	BQSquare	4.99%	4.00	3.64%	1.85
	RaceHorses	5.11%	2.23	4.63%	1.77
832x480	BasketballDrill	4.84%	2.15	4.26%	1.84
	BQMall	4.69%	2.32	4.10%	1.69
	RaceHorses	4.07%	1.73	3.53%	2.04
720P(1280x720)	FourPeople	4.39%	2.02	3.83%	1.65
	Johnny	5.55%	1.89	3.28%	1.54
	KristenAndSara	4.52%	1.76	3.21%	1.42
	Vidyo1	5.35%	2.01	3.87%	1.54
	Vidyo3	6.09%	2.25	4.54%	1.85
CIF(352x288)		3.94%	2.82	3.33%	1.79
416x240		4.82%	2.80	4.13%	1.81
832x480		4.53%	2.07	3.96%	1.86
720P(1280x720)		5.18%	1.99	3.75%	1.60
整体		4.70%	2.36	3.78%	1.74

表 5.6 训练帧数为 20 帧的阈值自适应快速转码算法性能

分辨率	序列	固定阈值		阈值自适应(k=20)	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	4.12%	2.52	3.38%	1.68
	container	4.21%	2.54	3.44%	1.87
	flower	3.48%	3.40	3.01%	1.88
416x240	BasketballPass	4.36%	2.16	4.03%	1.86
	BQSquare	4.99%	4.00	3.54%	1.94
	RaceHorses	5.11%	2.23	4.54%	1.77
832x480	BasketballDrill	4.84%	2.15	4.31%	1.69
	BQMall	4.69%	2.32	4.10%	1.64
	RaceHorses	4.07%	1.73	3.42%	2.04
720P(1280x720)	FourPeople	4.39%	2.02	3.80%	1.65
	Johnny	5.55%	1.89	3.19%	1.54
	KristenAndSara	4.52%	1.76	3.11%	1.39
	Vidyo1	5.35%	2.01	3.86%	1.54
	Vidyo3	6.09%	2.25	4.43%	1.85
CIF(352x288)		3.94%	2.82	3.28%	1.81
416x240		4.82%	2.80	4.04%	1.85
832x480		4.53%	2.07	3.94%	1.79
720P(1280x720)		5.18%	1.99	3.68%	1.59
整体		4.70%	2.36	3.73	1.74

表 5.7 各阈值快速转码算法的 RD 性能与速度性能对比表

	RD loss	加速比
固定阈值	4.70%	2.36
阈值自适应(k=5)	3.88%	1.88
阈值自适应(k=10)	3.82%	1.86
阈值自适应(k=15)	3.78%	1.74
阈值自适应(k=20)	3.73%	1.74

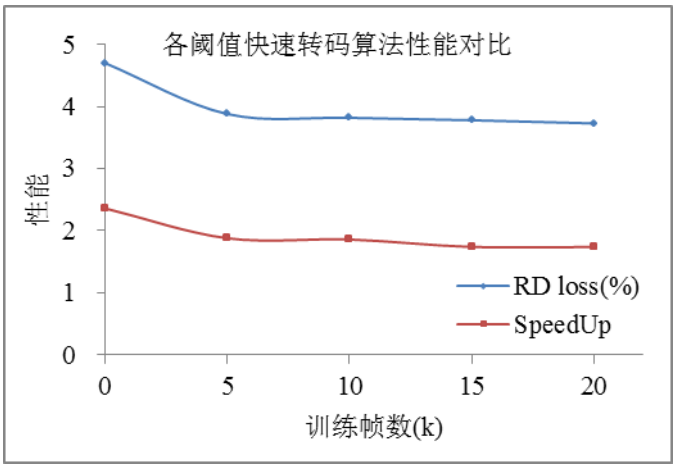


图 5.3 各阈值快速转码算法的 RD 性能与速度性能对比图

5.4.2 PU 快速模式决策算法性能

为验证 PU 快速模式决策算法的有效性，我们将该算法实验在全解全编转码的编码部分，并且，只应用于深度小于 2 的 CU 的模式决策中。表 5.8 展示了该算法的性能，其中 PU_FMD 表示在帧间加入 PU 快速模式决策算法的转码器。通过与全解全编的转码相比较，我们发现该算法在几乎不带来 RD 性能损失的同时，可以带来近 10%的转码加速。

此外，我们还进一步将该算法加入自适应阈值快速转码算法中，由于该算法只能进行 PART_2NxN、PART_Nx2N 及 AMP 模式的跳过决策，因此，它只适用于深度小于 2 且特征值介于 Tlow 与 Thigh 或特征值不可得的 CU 中。表 5.9 给出了加入 PU 快速模式决策算法的自适应阈值转码的性能，其中，AdaTH 代表训练帧数为 10，采用 Coeff_Num_Y 为特征的阈值自适应的快速转码算法的性能，AdaTH+PU_FMD 代表在 AdaTH 上加入 PU 快速模式决策

算法的转码性能。从表 5.9 可以看出，在自适应阈值转码中加入 PU 快速模式决策算法后，可以进一步的加速转码过程。然而， flower 序列及 BasketballPass 序列，AdaTH+PU_FMD 的转码速度反倒比 AdaTH 的转码速度低，这是因为，PU 快速模式决策算法自身会引入一定的编码复杂度，对于这几个序列，PU 快速模式决策带来的加速不足以抵消它引入的编码复杂度，从而导致速度不升反降。

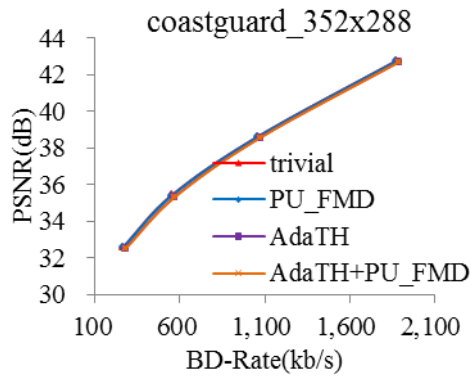
表 5.8 PU 快速模式决策算法性能

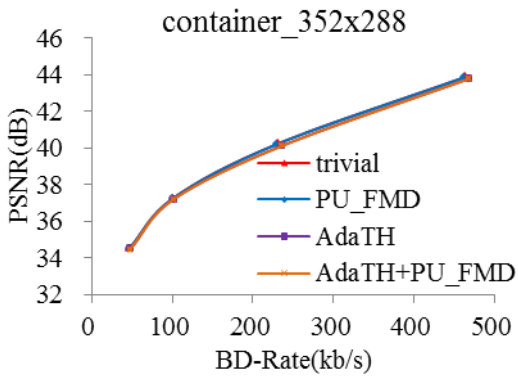
分辨率	序列	全解全编		PU_FMD	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	0.00%	1.00	0.15%	1.00
	container	0.00%	1.00	0.13%	1.11
	flower	0.00%	1.00	0.00%	1.06
416x240	BasketballPass	0.00%	1.00	0.12%	1.07
	BQSquare	0.00%	1.00	0.23%	1.08
	RaceHorses	0.00%	1.00	0.14%	1.00
832x480	BasketballDrill	0.00%	1.00	0.09%	1.18
	BQMall	0.00%	1.00	0.19%	1.13
	RaceHorses	0.00%	1.00	0.08%	1.13
720P(1280x720)	FourPeople	0.00%	1.00	0.37%	1.06
	Johnny	0.00%	1.00	0.80%	1.13
	KristenAndSara	0.00%	1.00	0.50%	1.03
	Vidyo1	0.00%	1.00	0.36%	1.13
	Vidyo3	0.00%	1.00	0.47%	1.32
CIF(352x288)		0.00%	1.00	0.09%	1.06
416x240		0.00%	1.00	0.16%	1.05
832x480		0.00%	1.00	0.12%	1.15
720P(1280x720)		0.00%	1.00	0.50%	1.13
整体		0.00%	1.00	0.26%	1.10

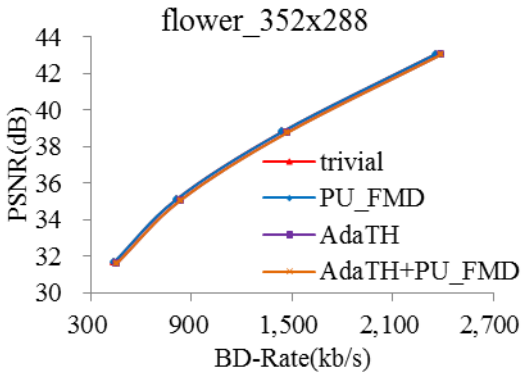
表 5.9 阈值自适应快速转码算法融合 PU 快速模式决策算法性能

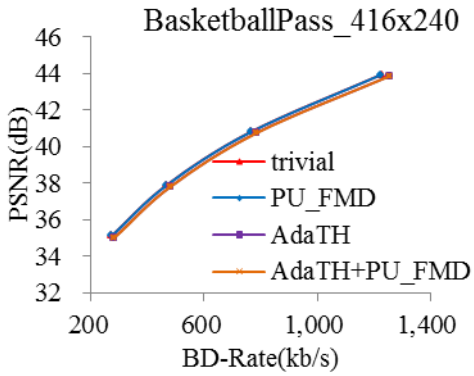
分辨率	序列	AdaTH(k=10)		AdaTH+PU_FMD	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	3.39%	1.92	3.67%	1.94
	container	3.57%	2.15	3.56%	2.18
	flower	3.11%	2.11	3.20%	2.04
416x240	BasketballPass	4.06%	2.04	4.21%	1.95
	BQSquare	3.67%	2.49	3.95%	2.60
	RaceHorses	4.68%	1.86	4.82%	1.95
832x480	BasketballDrill	4.33%	1.71	4.61%	2.01
	BQMall	4.13%	1.69	4.31%	1.82
	RaceHorses	3.57%	2.04	3.70%	2.04
720P(1280x720)	FourPeople	3.98%	1.69	4.19%	1.80
	Johnny	3.19%	1.57	4.14%	1.61
	KristenAndSara	3.41%	1.49	3.82%	1.68
	Vidyo1	3.89%	1.56	4.38%	1.85
	Vidyo3	4.47%	1.75	5.30%	2.03
CIF(352x288)		3.36%	2.06	3.48%	2.05
416x240		4.14%	2.13	4.33%	2.17
832x480		4.01%	1.81	4.21%	1.96
720P(1280x720)		3.79%	1.61	4.37%	1.79
整体		3.82%	1.86	4.13%	1.96

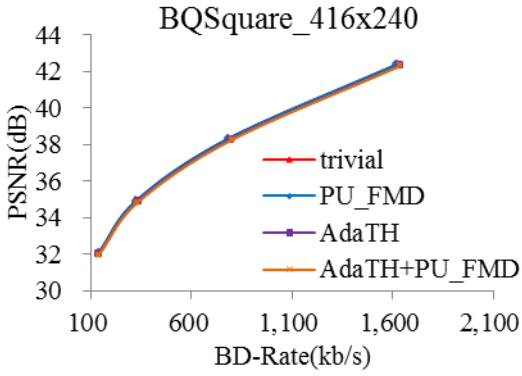
图 5.4 进一步展示了 PU 快速模式决策算法的编码 RD 曲线。其中，红线带表全解全编的转码方式，蓝线代表编码部分加入 PU 快速模式决策算法的转码方式，紫线代表阈值自适应的快速转码算法的性能，橙线代表阈值自适应的快速转码加入 PU 快速模式决策算法的转码性能。从图中可看出，PU 快速模式决策算法的 RD 曲线几乎与全解全编转码的 RD 曲线融合。后两者也十分接近于全解全编的转码 RD 曲线。

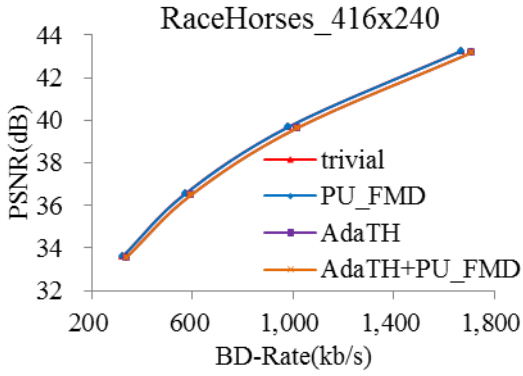

coastguard_352x288
PSNR(dB)
BD-Rate(kb/s)
trivial
PU_FMD
AdaTH
AdaTH+PU_FMD

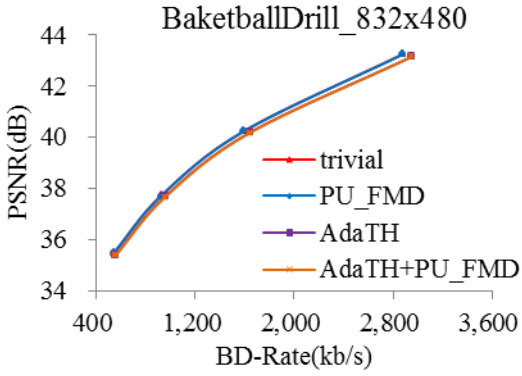

container_352x288
PSNR(dB)
BD-Rate(kb/s)
trivial
PU_FMD
AdaTH
AdaTH+PU_FMD

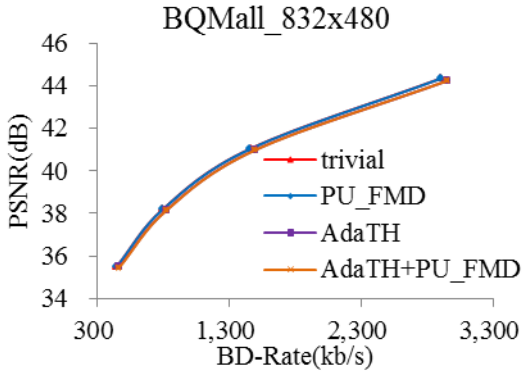

flower_352x288
PSNR(dB)
BD-Rate(kb/s)
trivial
PU_FMD
AdaTH
AdaTH+PU_FMD


BasketballPass_416x240
PSNR(dB)
BD-Rate(kb/s)
trivial
PU_FMD
AdaTH
AdaTH+PU_FMD


BQSquare_416x240
PSNR(dB)
BD-Rate(kb/s)
trivial
PU_FMD
AdaTH
AdaTH+PU_FMD


RaceHorses_416x240
PSNR(dB)
BD-Rate(kb/s)
trivial
PU_FMD
AdaTH
AdaTH+PU_FMD


BaketballDrill_832x480
PSNR(dB)
BD-Rate(kb/s)
trivial
PU_FMD
AdaTH
AdaTH+PU_FMD


BQMall_832x480
PSNR(dB)
BD-Rate(kb/s)
trivial
PU_FMD
AdaTH
AdaTH+PU_FMD

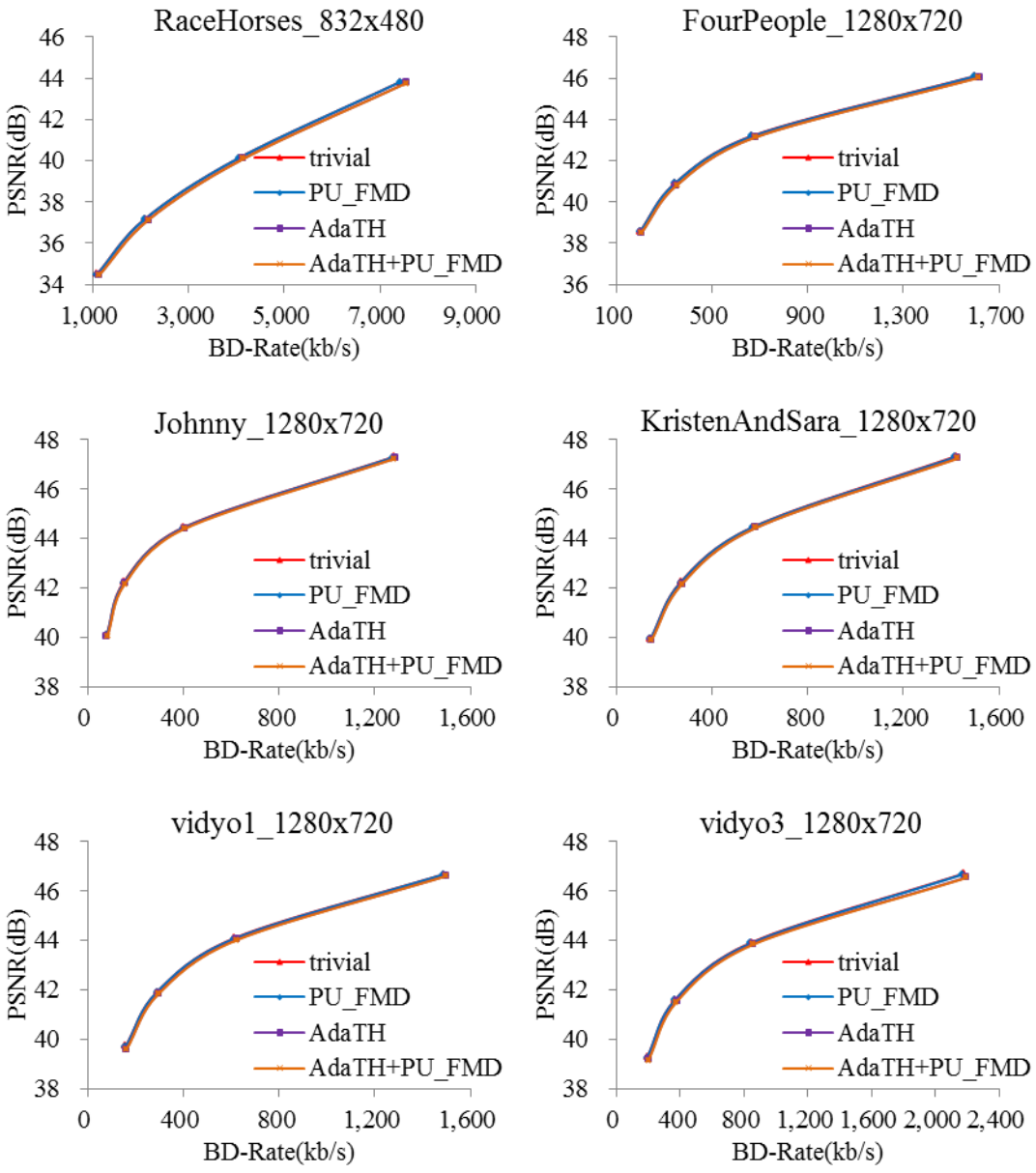


图 5.4 各快速转码算法的 RD 曲线

5.4.3 整体算法性能对比

为进一步验证本算法的有效性，我们进一步实现了文献[66]提出的快速转码算法，本章提出的阈值自适应快速转码算法与文献[66]的算法的性能总结至表 5.10 中。其中，AdaTH+PU_FMD 显示了将快速 PU 模式决策算法嵌入到阈值自适应快速转码算法的性能，其中训练帧数为 10 帧。从表中可以看出，平均情况下，AdaTH+PU_FMD 的 RD 损失小于文献[66]的 RD 损失，但转码速度略高于文献[66]的转码速度。由此可以看出，文章提出的快速转码算法十

分有效，适用于对转码 RD 性能有高要求的转码应用。

表 5.10 阈值自适应快速转码算法性能与文献[66]性能对比

分辨率	序列	AdaTH(k=10)+PU_FMD		文献[66]	
		BD-Rate	加速	BD-Rate	加速
CIF(352x288)	coastguard	3.67%	1.94	4.20%	1.67
	container	3.56%	2.18	4.73%	2.06
	flower	3.20%	2.04	3.46%	1.93
416x240	BasketballPass	4.21%	1.95	5.04%	1.70
	BQSquare	3.95%	2.60	4.99%	2.28
	RaceHorses	4.82%	1.95	4.75%	1.69
832x480	BasketballDrill	4.61%	2.01	5.13%	1.82
	BQMall	4.31%	1.82	5.16%	1.64
	RaceHorses	3.70%	2.04	3.80%	1.83
720P(1280x720)	FourPeople	4.19%	1.80	4.75%	1.62
	Johnny	4.14%	1.61	3.71%	1.50
	KristenAndSara	3.82%	1.68	4.17%	1.42
	Vidyo1	4.38%	1.85	4.97%	1.54
	Vidyo3	5.30%	2.03	5.07%	1.85
CIF(352x288)		3.48%	2.05	4.13%	1.89
416x240		4.33%	2.17	4.93%	1.89
832x480		4.21%	1.96	4.70%	1.76
720P(1280x720)		4.37%	1.79	4.53%	1.59
整体		4.13%	1.96	4.57%	1.75

5.5 本章小结

本章提出一种阈值自适应的快速转码算法，它可以弥补固定阈值下所有序列使用相同阈值而导致阈值不准确的问题。通过在线训练方式，本算法将

自适应调整阈值以获取适用于不同特征、不同 QP 的序列的转码。实验结果表明，阈值自适应的快速转码算法将带来更高效的转码 RD 性能，然而，由于每次转码时需进行在线阈值训练，因此，本算法的速度性能略低于固定阈值转码算法。此外，实验还进一步研究了不同训练帧数下的阈值自适应转码的 RD 性能与速度性能，结果表明，随着训练帧数的增加，RD 性能增加，但速度性能下降。除阈值自适应的快速转码算法外，本章还提出一种 PU 快速模式决策算法以进一步加速转码过程。实验表明，该快速模式决策算法在几乎不损失 RD 性能的同时，可以进行转码加速。然而，由于该算法只适应于深度小于 2 且特征值介于 Tlow 与 Thigh 或特征值不可得的 CU 中，因此，加速比较有限。

第 6 章 总结与展望

6.1 工作总结

随着 HEVC 标准的制定形成，H.264 至 HEVC 的快速转码算法研究已经成为标准间转码研究的新热点。本文就该课题提出了三套解决方案：1、基于 AMV-RDO 的快速转码算法，该算法利用解码的 H.264 运动矢量信息来进行当前 CTU 的四叉树划分决策与 PU 的预测，并进一步利用 HEVC 的原始率失真模型及 H.264 的模式信息进行 CTU 四叉树叶子结点的 PU 决策，从而生成 HEVC 码流；2、固定阈值下的快速转码算法，它是一种基于统计的快速转码方式，利用当前 CU 的特征值与阈值之间的关系进行当前 CU 的划分决策与 PU 选择，从而进行转码加速；3、阈值自适应的快速转码算法，它为弥补固定阈值下所有序列使用相同阈值而导致阈值不准确而提出的快速转码算法，通过在线训练方式，该算法将自适应调整阈值以获取适用于不同特征、不同 QP 的序列的转码，此外，还提出一种 PU 快速模式决策算法，以进一步的加速转码过程。这三套算法的转码性能各不相同，实验结果表明：1、基于 AMV-RDO 的快速转码算法在引入一定的质量损失的前提下，具有非常高的加速比，相比于全解全编的转码，本算法的加速比达到 7.61 倍，即节省了 87% 的转码时间，因此，本算法十分适用于对质量要求并非十分严格，但具有实时要求的转码应用中；2、固定阈值下的快速转码算法虽然不能带来与基于 AMV-RDO 的快速转码算法相同的转码速度，但它的转码性能非常好，较全解全编转码在几乎不带来性能损失的同时，转码速度提升 1.36 倍，因此，该算法适用于对转码 RD 性能具有高要求的转码应用中；3、阈值自适应的快速转码算法由于可以自适应的调整阈值，因此，RD 性能比固定阈值更高，但转码速度比固定阈值转码略低，它在带来更小 RD 性能损失的同时，转码速度提升一倍；4、相比于文献[66]提出的快速转码算法，基于 AMV-RDO 的快速转码算法尽管 RD 损失较前者大，但转码速度远快于文献[66]的转码速度，固定阈值下的快速转码算法与阈值自适应的快速转码算法的 RD 性能均优于文献[66]提出的转码算法，且转码速度也略高于文献[66]的转码速度。

综上所述，本文所研究的三类快速转码算法性能各不相同：基于 AMV-RDO 的快速转码算法具有十分高效的转码速度性能，但将引入一定的转码 RD 性能损失；阈值自适应的快速转码算法的转码 RD 性能最优，但转

码速度也最低，它在几乎不带来 RD 性能损失的同时，较全解全编转码可以提升一倍的转码速度；固定阈值的快速转码算法的 RD 性能及速度性能均介于基于 AMV-RDO 的快速转码算法及阈值自适应的快速转码算法。因此，这三种快速转码算法可以应用于不同的应用场景：对转码 RD 性能要求不高，但具有严格实时要求的转码应用中，可以应用基于 AMV-RDO 的快速转码算法；对性能要求严格，但转码速度要求不高的转码应用中，可以应用阈值自适应的快速转码算法；对性能要求严格，但对内存要求也很高的转码应用中，则可以应用固定阈值的快速转码算法，以避免阈值自适应训练过程中所消耗的大量内存。

6.2 未来展望

本文提出的三套快速转码算法方案均为理论研究工作，但转码技术是应用性非常强的一项技术，因此，在将这些方案进行实际应用中，必须还存在很多的优化工作。因此，在未来的工作中，希望能够在实际应用条件下对本文的研究内容进行更多的验证和测试，甚至进一步改进算法。此外，随着计算机的处理能力逐渐增强，计算并行也是一项新的研究热点，为使快速转码更快速，如何结合快速转码技术与并行技术也将是一项新的研究热点。

参考文献

[1] “Generic Coding of Moving Pictures and Associated Audio Information – Part 2: Video,” ITU-T and ISO/IEC JTC 1, ITU-T Recommendation H.262 and ISO/IEC 13818-2(MPEG-2), 1994.

[2] ITU-T and ISO/IEC JTC 1, “Advanced Video Coding for Generic Audio-Visual Services,” ITU-T Recommendation H.264 and ISO/IEC 14496-10 (AVC), version 1, 2003, version 2, 2004, versions 3, 4, 2005, versions 5, 6, 2006, versions 7, 8, 2007, versions 9, 10, 11, 2009, versions 12, 13, 2010, versions 14, 15, 2011, version 16, 2012.

[3] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[4] B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan, Y.-K. Wang, T. Wiegand, “High Efficiency Video Coding (HEVC) text specification draft 10 (for FDIS & Consent)” ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document, JCTVC-L1003, Jan. 2013.

[5] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1648–1667, Dec. 2012.

[6] T. Wedi, “Motion compensation in H.264/AVC,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 577–586, July 2003.

[7] T. Wiegand, X. Zhang and B. Girod, “Long-term memory motion-compensated prediction,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, pp. 70–84, Feb. 1999.

[8] T. Wiegand and B. Girod, “Multi-Frame Motion-Compensated Prediction for Video Transmission”. Norwell, MA: Kluwer, 2001.

[9] M. Flierl and B. Girod, “Generalized B pictures and the draft JVT/H.264 video compression standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 587–597, July 2003.

[10] 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*, vol. 13, pp. 598–603, July 2003.

[11] D. Marpe, H. Schwarz, and T. Wiegand, “Context-adaptive binary arithmetic coding in the H.264/AVC video compression standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 620–636, July 2003.

[12] P. List, A. Joch, J. Lainema, G. Bjøntegaard, and M. Karczewicz, “Adaptive deblocking filter,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp.

614–619, July 2003.

[13]I.-K. Kim, J. Min, T. Lee, W.-J. Han, and J. Park, “Block partitioning structure in the HEVC standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1697–1706, Dec. 2012.

[14]J. Lainema, F. Bossen, W. J. Han, J. Min and K. Ugur, “Intra Coding of the HEVC Standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1792-1801, Dec. 2012.

[15]P. Helle, S. Oudin, B. Bross, D. Marpe, M. O. Bici, K. Ugur, J. Jung, G. Clare and T. Wiegand, “Block Merging for Quadtree-Based Partitioning in HEVC”, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1720-1731, Dec. 2012.

[16]J.L. Lin, Y.W. Chen, Y.P. Tsai, Y.W. Huang, and S. Lei, “Motion vector coding techniques for HEVC,” *In IEEE International Workshop on Multimedia Signal Processing*, pages 1-6, 2011.

[17]T. Nguyen, P. Helle, M. Winken, B. Bross, D. Marpe, H. Schwarz, and T. Wiegand, “Transform coding techniques in hevc,” *Selected Topics in Signal Processing, IEEE Journal of* , vol. 7, no. 6, pp. 978–989, Dec 2013.

[18]V. Sze and M. Budagavi, “High throughput CABAC entropy coding in HEVC,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1778–1791, Dec. 2012.

[19]C.-M. Fu et al., “Sample adaptive offset in the HEVC standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1755–1764, Dec. 2012.

[20]J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan and T. Wiegand, “Comparison of the Coding Efficiency of Video Coding Standards–Including High Efficiency Video Coding (HEVC)”, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1669-1684, Dec. 2012.

[21]A. Vetro, C. Christopulos, and H. Sun, “Video transcoding architectures and techniques: An overview,” *IEEE Signal Processing Magazine*, vol. 20, no. 2, pp. 18–29, Mar. 2003.

[22]J. Xin, C.-W. Lin, and M.-T. Sun, “Digital video transcoding,” *Proceeding of The IEEE*, vol. 93, no. 1, pp. 84–97, Jan. 2005.

[23]I. Ahmad, X. Wei, Y. Sun, and Y.-Q. Zhang, “Video transcoding: an overview of various techniques and research issues,” *IEEE Transactions on Multimedia*, vol. 7, no. 5, pp. 793–804, Oct. 2005.

[24]A. Eleftheriadis and D. Anastassiou, “Constrained and general dynamic rate shaping of compressed digital video,” in *Proc. IEEE Int. Conf. Image Processing*, vol. 3, 1995, pp. 396–399.

[25]H. Sun, W. Kwok, and J. W. Zdepski, “Architectures for MPEG compressed bitstream scaling,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 2, pp. 191–199, Apr. 1996.

[26]Y. Nakajima, H. Hori, and T. Kanoh, “Rate conversion of MPEG coded video by

re-quantization process," in *Proc. IEEE Int. Conf. Image Processing*, vol. 3, 1995, pp. 408–411.

[27]J. Youn, M.-T. Sun, and J. Xin, "Video transcoder architectures for bit rate scaling of H.263 bit streams," in *Proc. ACM Multimedia*, Nov. 1999, pp. 243–250.

[28]D. G. Morrison, M. E. Nilson, and M. Ghanbari, "Reduction of the bit-rate of compressed video while in its coded form," in *Proc. 6th Int. Workshop Packet Video*, 1994, pp. D17.1–D17.4.

[29]G. Keesman, R. Hellinghuizen, F. Hoeksema, and G. Heideman, "Transcoding of MPEG bitstreams," *Signal Process. Image Commun.*, vol. 8, no. 6, pp. 481–500, Sep. 1996.

[30]P. A. A. Assuncao and M. Ghanbari, "A frequency-domain video transcoder for dynamic bitrate reduction of MPEG-2 bit streams," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 8, pp. 953–967, Dec. 1998.

[31]Jeongnam Youn; Ming-Ting Sun; Chia-Wen Lin; "Motion vector refinement for high-performance transcoding," *Multimedia, IEEE Transactions on* , vol.1, no.1, pp.30-40, Mar 1999.

[32]N. Merhav and V. Bhaskaran, "Fast algorithms for DCT-domain image downsampling and for inverse motion compensation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, no. 3, pp. 468–476, Jun. 1997.

[33]J. Song and B.-L. Yeo, "A fast algorithm for DCT-domain inverse motion compensation based on shared information in a macroblock," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, no. 5, pp. 767–775, Aug. 2000.

[34]C.-W. Lin and Y.-R. Lee, "Fast algorithms for DCT-domain video transcoding," in *Proc. IEEE Int. Conf. Image Processing*, vol. 1, 2001, pp. 421–424.

[35]S. Acharya and B. Smith, "Compressed domain transcoding of, MPEG," in *Proc. IEEE Int. Conf. Multimedia Computing and Systems*, 1998, pp. 295–304.

[36]S. Liu and A. C. Bovik, "Local bandwidth constrained fast inverse motion compensation for DCT-domain video transcoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 5, pp. 309–319, May 2000.

[37]J.-N. Hwang and T.-D. Wu, "Motion vector re-estimation and dynamic frame-skipping for video transcoding," in *Conf. Rec. 32nd Asilomar Conf. Signals, System & Computer*, vol. 2, 1998, pp. 1606–1610.

[38]J. Youn, M.-T. Sun, and C.-W. Lin, "Motion vector refinement for high-performance transcoding," *IEEE Trans. Multimedia*, vol. 1, no. 1, pp. 30–40, Mar. 1999.

[39]T. Shanableh and M. Ghanbari, "Heterogeneous video transcoding to lower spatial-temporal resolutions and different encoding formats," *IEEE Trans. Multimedia*, vol. 2, no. 2, pp. 101–110, Jun. 2000.

[40]M.-J. Chen, M.-C. Chu, and C.-W. Pan, "Efficient motion-estimation algorithm for reduced frame-rate video transcoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12,

no. 4, pp. 269–275, Apr. 2002.

[41]N. Bjork and C. Christopoulos, “Transcoder architecture for video coding,” *IEEE Trans. Consumer Electron.*, vol. 44, pp. 88–98, Feb. 1998.

[42]B. Shen, I. K. Sethi, and B. Vasudev, “Adaptive motion-vector resampling for compressed video down scaling,” *IEEE Trans. Circuits Syst.Video Technol.*, vol. 9, no. 6, pp. 929–936, Sep. 1999.

[43]P. Yin and M. Wu, “Video transcoding by reducing spatial resolution,” in *Proc. IEEE Int. Conf. Image Processing*, vol. 1, 2000, pp. 972–975.

[44]G. Shen, B. Zeng, Y.-Q. Zhang, and M. L. Liou, “Transcoder with arbitrarily resizing capability,” in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS) 2001*, vol. 5, Sydney, NSW, Australia, May 2001, pp. 25–28.

[45]J. Xin, M.-T. Sun, K. Chun, and B. S. Choi, “Motion re-estimation for HDTV to SDTV transcoding,” in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS) 2002*, vol. 4, Geneva, Switzerland, May 2002, pp. 715–718.

[46]K. Takahashi, K. Satoh, T. Suzuki, and Y. Yagasaki, “Motion vector synthesis algorithm for MPEG2-to-MPEG4 transcoder,” in , *Proc. SPIE—Visual Communications and Image Processing*, vol. 4310, San Jose, CA, Jan. 2001, pp. 872–882.

[47]Moiron, S.; Ghanbari, M.; "Reduced complexity intra mode decision for resolution reduction on H.264 transcoders," *Consumer Electronics, IEEE Transactions on* , vol.55, no.2, pp.606-612, May 2009

[48]P. Zhang, Y. Lu, Q. Huang, and W. Gao, “Mode mapping method for H.264 spatial downscaling transcoding,” in 2004 International Conference on Image Processing, 2004. ICIP ’04. IEEE, 2004, pp. 2781–2784.

[49]C.-H. Li, C.-N. Wang, and T. Chiang, “A fast downsizing video transcoder based on H.264 standard,” in Advances in Multimedia Information Processing - PCM 2004, 2005, pp. 215–223.

[50]H. Kalva and B. Petljanski, "Exploiting the directional features in MPEG-2 for H.264 intra transcoding," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 2, pp.706 -711 2006.

[51]X. Lu, A. M. Tourapis, P. Yin and J. Boyce, "Fast mode decision and motion estimation for H.264 with a focus on MPEG-2/H.264 transcoding," *Pro. 2005 IEEE Int. Symp. Circuits Syst.*, pp.1246 -1249 2005.

[52]Y. Su, J. Xin, A. Vetro and H. Sun, "Efficient MPEG-2 to H.264/AVC intra-transcoding in transform-domain," *Proc. IEEE Int. Symp. Circuits Syst.*, pp.1234 -1237 2005.

[53]Z. Zhou, S. Sun, S. Lei and M. T. Sun, "Motion information and coding mode reuse for MPEG-2 to H.264 transcoding," *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, pp.1230 -1233 2005.

[54]G. Fernndez-Escribano, H. Kalva, P. Cuenca, L. Orozco-Barbosa and A. Garrido, "A fast

MB mode decision algorithm for MPEG-2 to H.264 P-frame transcoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, pp.172 -185 2008.

[55]Liang Fan, Siwei Ma and Feng Wu "overview of AVS Video Standard," IEEE International Conference on Multimedia and Expo(ICME), 2004.

[56]GB/T 20090.2-2006. 信息技术 先进音视频编码—第2部分：视频.中国：中国国家标准化管理委员会，中华人民工共和国国家质量监督检验检疫总局，2006.

[57]Yu L, Yi F, Dong J, Zhang C, "Overview of AVS-video: tools, performance and complexity," in *Proc. SPIE.*, pp 679–690, 2005.

[58]Wang X-F, Zhao D-B, "Performance comparison of AVS and H.264/AVC video coding standards," *J Comput Sci. Technol.*, pp 310–314, 2006.

[59]Z. H. Wang, X. Y. Ji, and W. Gao, "Effective algorithms for fast transcoding of AVS to H.264/AVC in the spatial domain," *Journal Multimedia Tools and Applications Springer Netherlands*, pp 175-202, 2007.

[60]Wang Z, Gao W, Zhao D, Huang Q, "A fast intra mode decision algorithm for AVS to H.264 transcoding," In *International Conference on Multimedia & Expo*, pp 61–64, 2006.

[61]尚凯, 张万绪, "AVS-H.264视频转码快速算法," *计算机工程*, pp 234- 244, 2010.

[62]B. G. Wang, Y. H. Shi, and B. C. Yin, "Transcoding of H.264 bitstream to AVS bitsteam," Wireless Communications, Networking and Mobile Computing, pp 9: 1-4, 2009.

[63]W. Jiang, et al., "Fast transcoding from H. 264 to HEVC based on region feature analysis," *Multimedia Tools and Applications*, pp. 1-22, 2013.

[64]D. Zhang , B. Li , J. Xu and H. Li "Fast transcoding from H.264 AVC to highefficiency video coding", *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, pp.651 -656, 2012.

[65]E. Peixoto and E. Izquierdo, "A complexity-scalable transcoder from H.264/AVC to the new HEVC codec," *Proc. IEEE Int. Conf. Image Process(ICIP)*, pp.737 -740 2012.

[66]E. Peixoto, T. Shanableh, and E. Izquierdo, "H.264/AVC to HEVC video transcoder based on dynamic thresholding and content modeling," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. PP, no. 99, pp. 1-1, 2013.

[67]T. Shen, Y. Lu, Z. Wen, L. Zou, Y. Chen, and J. Wen, "Ultra fast h.264/avc to hevc transcoder," in *Proceedings of the 2013 Data Compression Conference (DCC 2013)*, pp. 241-250, March 2013.

[68]A. J. Diaz-Honrubia, J. L. Martinez, J. M. Puerta, "Fast quadtree level decision algorithm for H.264/HEVC transcoder," in IEEE International Conference on Image Processing (ICIP), 2014.

[69]E. Peixoto, B. Macchiavello, E. M. Hung, A. Zaghetto, T. Shanableh, and E. Izquierdo, "An H.264/AVC to HEVC video transcoder based on mode mapping," in IEEE International Conference on Image Processing (ICIP), pp. 1972-1976, September 2013.

[70]E. Peixoto, B. Macchiavello, E. M. Hung, "Fast H.264/AVC to HEVC transcoding based

on machine learning," in IEEE International Telecommunications Symposium (ITS), 2014.

[71]Zong-Yi Chen, Jiunn-Tsair Fang, Tsai-Ling Liao and Pao-Chi Chang, "Efficient PU mode decision and motion estimation for H.264/AVC to HEVC transcoder," *Signal & Image Processing: An International Journal (SIPIJ)*, vol. 5, No. 2, April 2014.

[72]H. Shen, X. Sun, and F. Wu, "Fast H.264/MPEG-4 AVC Transcoding Using Power-Spectrum Based Rate-Distortion Optimization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 6, pp. 746–755, June 2008.

[73]Peiyin Xing, Yonghong Tian, Xianguo Zhang, Yaowei Wang and Tiejun Huang, "A coding unit classification based AVC-to-HEVC transcoding with background modeling for surveillance videos," *Visual Communications and Image Processing (VCIP)*, pp.1-6, Nov. 2013.

[74]司俊俊, 面向 HEVC 的码率控制研究, 北京大学硕士学位论文, 2014.06.

[75]H.264/AVC JM Reference Software is publicly available at: http://iphome.hhi.de/suehring/tml/.

[76]G. Fernandez-Escribano, H. Kalva, J. Martinez, P. Cuenca, L. OrozcoBarbosa, and A. Garrido, "An MPEG-2 to H.264 video transcoder in the baseline profile," IEEE Trans. Circuits Syst. Video Technol., vol. 20, no. 5, pp. 763–768, May 2010.

[77]C. Holder, T. Pin, and H. Kalva, "Improved machine learning techniques for low complexity MPEG-2 to H.264 transcoding using optimized codecs," in Proc. Int. Conf. Consumer Electron. (ICCE), Jan. 2009, pp. 1–2.

攻读硕士学位期间的科研成果

会议论文

[1] **Lei Chen**, Ronggang Wang, Siwei Ma, "Low-cost multi-hypothesis motion compensation for video coding," SPIE Visual Information Processing and Communication V, 2014.

[2] **Lei Chen**, Ronggang Wang, Siwei Ma, “Tagged multi-hypothesis motion compensation scheme for video coding,” IEEE Visual Communications and Image Processing (VCIP), 2014.

标准提案

[1] **陈蕾**，余琴，马思伟，“关于SAO在码流中的标识，” AVS，M3230，深圳，2013, 12.

[2] **陈蕾**，司俊俊，王苫社，“在LDP配置下修改参考帧管理中的QPoffset值及lambda值,”AVS, M3410，大连, 2014, 06.

[3] **Lei Chen**, Ronggang Wang, “Multi-hypothesis motion compensation for P frame,” ISO/IEC JTC1/SC29/WG11, NXXX, Vienna, July, 2013.

[4] **Lei Chen**, Ronggang Wang, “New B frame encoding technology for internet video coding,” ISO/IEC JTC1/SC29/WG11, M27966, Geneve, Jan, 2013.

专利申请

[1] **陈蕾**，马思伟，“一种针对屏幕视频帧间残差的基础色索引映射算法,”201410188957.3

[2] **陈蕾**，王荣刚，”一种基于P帧的多假设运动补偿方法,” DHC1310230PC.

[3] **陈蕾**，王荣刚，“一种基于P帧的多假设运动补偿编码方法,” DHC1310231PC.

所获奖励

2014年北京大学中营奖学金；
2014年北京大学三好学生；
2013年信息工程学院科研优秀奖；

致谢

犹记得四年前，当我第一次踏入实验室的时候，仍保有一张稚气的脸。时间一晃而过，四年时间太匆匆。值此毕业之际，想感谢的人太多。

首先，我要感谢我的导师马思伟老师。四年前，是马老师带领我走入视频编码的世界，从学习入门知识到进行学术研究，我的每一步成长，都离不开马老师的谆谆教诲。您广博的知识、刻苦的精神以及不断追求的勇气一直鞭挞着我不断前行。

其次，我要感谢深圳研究生院的王荣刚老师，在深圳的一年时间，是您让我对编码领域有了进一步的认识，让我对自己的能力有了信心。感谢深圳研究生院的董胜富老师、王振宇老师、李英老师，感谢深圳研究生院的辛柏成师兄，万杰师兄，感谢我深圳研究生院的同学们对我深圳这一年的无私帮助与鼓励。感谢深圳研究生院的韩冰杰同学、李旭峰同学、焦剑波同学、丁磊同学、吕正光同学、张艺同学、张建龙同学、陈钦水同学这一年来的共同学习与帮助。

感谢数字媒体所的高文老师、黄铁军老师、王亦洲老师、田永鸿老师、解晓东老师、熊瑞勤老师、段凌宇老师、蒋婷婷老师、贾惠柱老师对我的指导。感谢数字媒体所的蔡光辉老师、董巍老师、刘敬老师、李伟老师在我们学习及生活上的帮助。

感谢视频编码组的赵欣师兄、王悦师兄、张新峰师兄、王诗淇师兄、王苫社师兄、赵亮师兄、齐峰师兄、张健师兄、赵琛师姐、林松师兄、谭颂超师兄、刘航帆师兄、司俊俊师姐、余琴师姐、蔡砚刚师兄、宋治海师兄等的无私帮助。

感谢视频编码组的同学及师弟师妹们：马俊铖、师圣、姜晓龙、马莉、穆晶、范娟婷、罗法蕾、张翔、谢利娟、王钊、赵磊、李嘉豪等对我的支持与帮助。

我要由衷的感谢我的父母，在我脆弱、不坚强的时候，永远的支持我，给予我信念与力量，帮助我完成硕士生涯。

最后，感谢自己，坚持走到今天。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　日期：　　年　　月　　日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校□一年/□两年/□三年以后，在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：　　导师签名：

日期：　　年　　月　　日